Tampere University

Arttu Hautakoski

AARIA: A SIMULATION FRAMEWORK OF RECONFIGURABLE MANIPULATORS FOR DEEP LEARNING SCENARIOS

Faculty of Engineering and Natural Sciences
Master's Thesis
March 2019

ABSTRACT

Arttu Hautakoski: Aaria: A Simulation Framework of Reconfigurable Manipulators for Deep Learning Scenarios
Master's Thesis
Tampere University
Master's Degree Programme in Automation
Examiners: Prof. Jouni Mattila, Ph.D. Mohammad Mohammadi Aref
March 2019

This thesis presents a simulation platform called Aaria. The purpose of Aaria is the generation of synthetic movement data for machine learning applications in robotics. The goal is to learn deep features that are common to the robotic structures so that the solution could generalize to all robots regardless of mass or structure. This thesis also discusses the literature about reconfigurable mechanisms, system parameter identification, human activity recognition and synthetic data. The descriptions of the components of a convolutional neural network are also included in this work along with the network architecture used in the presented machine-learning task.

The modular structure of Aaria allows it to generate any kind of open chain manipulator with a maximum of six degrees of freedom defined by modified Denavit-Hartenberg parameters. One of the advantages of Aaria is its ability to generate randomized structures and thus generate a wide variety of time series data. Time series data can be considered as one-dimensional images, which makes them a suitable data type for convolutional neural networks. Multiple time series can form a two-dimensional structure similar to images. This kind of two-dimensional time series dataset can be used as training data for learning deep features of robotic structures. In addition to being able to generate random strictures, Aaria can also be used to simulate and gather data from specific structures.

Some of the generated synthetic data was used in a machine-learning task to estimate the lengths and masses of swinging structures based on input torques and rotation angles, velocities and accelerations. The results were promising with 2.5 % mean relative error for both length and mass.

Keywords: machine-learning, synthetic data, robot

TIIVISTELMÄ

Arttu Hautakoski: Aaria: A Simulation Framework of Reconfigurable Manipulators for Deep Learning Scenarios
Diplomityö
Tampereen yliopisto
Automaatiotekniikan tutkinto-ohjelma
Tarkastajat: Prof. Jouni Mattila, TkT. Mohammad Mohammadi Aref
Maaliskuu 2019

Tässä työssä esitellään Aaria-simulaattori. Aarian tarkoitus on tuottaa synteettistä liikedataa robotiikan koneoppimis-sovellutuksiin. Tavoitteena on oppia roboteille yhteisiä syviä ominaisuuksia niin, että tuloksia voidaan soveltaa kaikkiin robotteihin massasta ja rakenteesta riippumatta. Tässä työssä käsitellään myös aiheeseen liittyvää kirjallisuutta uudelleenkonfiguroitavista mekanismeista, järjestelmän parametrien identifioinnista, ihmisen toiminnan tunnistamisesta ja synteettisestä datasta. Lisäksi tässä työssä esitellään konvoluutioneuroverkkojen toimintaa ja niiden avulla toteutetun, tässä työssä esitellyn koneoppimis-sovelluksen arkkitehtuuria.

Aarian modulaarinen rakenne mahdollistaa minkä tahansa maksimissaan kuuden vapausasteen robotin simuloimista modifioitujen Denavit-Hartenberg parametrien perusteella. Yksi Aarian eduista on sen kyky simuloida satunnaisia rakenteita ja siten tuottaa paljon rikasta ja laadukasta aikasarjadataa. Aikasarjojen voidaan ajatella olevan yksiulotteisia kuvia, mikä tekee niistä konvoluutioneuroverkoille soveltuvaa dataa. Useita aikasarjoja voidaan yhdistää kuvien kaltaisiksi kaksiulotteisiksi rakenteiksi. Tällaisia kaksiulotteisia aikasarjoja voidaan käyttää opetusdatana robottien syvien ominaisuuksien oppimisessa. Satunnaisten rakenteiden simuloinnin lisäksi Aariaa voidaan käyttää jonkin tietyn rakenteen simuloimiseen.

Tuotettua synteettistä dataa käytettiin koneoppimis-sovelluksessa, jossa liikkuvien puomien pituuksia ja massoja arvioitiin vääntömomenttien, kääntökulmien, nopeuksien ja kiihtyvyyksien perusteella. Sovelluksen tuloksena saatiin puomien keskimääräiseksi suhteelliseksi pituus- ja massavirheeksi 2.5 %.

Avainsanat: Koneoppiminen, synteettinen data, robotti

PREFACE

I want to thank my examiners Mohammad Mohammadi Aref and Jouni Mattila for helpful insights regarding the content of the thesis. I want thank my family for all the support and encouragement they have given me throughout my studies. Special thanks for my mother for reminding me to focus on the important things. Thanks to Terhi Keskinen for guiding me to study automation. I also want to thank my friends for all the help in studies and projects during the past few years.

In Tampere, Finland, on 20.03.2019

Arttu Hautakoski

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
DH	Denavit-Hartenberg
DL	Deep Learning
DOF	Degrees of Freedom
IMU	Inertial Measurement Unit
MAE	Mean Absolute Error
ML	Machine Learning
ReLU	Rectified Linear Unit

1. INTRODUCTION

1.1 Motivation

Deep learning (DL), machine learning (ML) and artificial intelligence (AI) have become popular in various applications over the past few years. These terms can often be seen used interchangeably but they are not synonyms. AI is defined as the science of making intelligent machines (McCarthy 2007). ML can be defined as a scientific field within AI that studies computer systems that learn with experience and the learning process itself (Mitchell 2006). DL is a subset of ML that learns complicated concepts by combining simpler concepts (Goodfellow 2016).

According to Gartner (2018), DL is at the peak of the hype cycle meaning that the general population's interest and expectations are at all-time high and they are about to start declining. This only means that it will be talked about less but the technologies are here to stay. Many companies are looking into ML and AI hoping to find solutions for their problems. They may find a similar application that they were looking for, which means that their problem can most likely be solved with ML. Even if a similar application does not yet exist, it can be made. In both cases, the biggest obstacle between the application needs and an ML functionality is training data and its quality.

ML often needs big data. Big data is defined as data that has high volume, velocity and variety (Gartner 2019). The application area or the ML method does not change the fact that training a working ML model requires a large amount of data and a well performing model requires even more data. The training data cannot be just any data; it needs to be relevant to the task, intact and labeled. For example, if the goal is to use ML to recognize different products on a conveyor belt, its data has to include pictures of all the parts that will appear on the conveyor belt. In addition, the pictures have to be clear, the parts have to be fully visible, and of course, the pictures must not be corrupted in any way. The quality of the data depends on its annotation as well. Each picture must be accompanied with information about what is in that picture or what the ML should give as the answer when it receives that picture. Collecting a large amount of data that has these qualities can be very expensive and time consuming. Sometimes collecting data may not be reasonable option at all. In cases like these, the options are to give up on the ML solution for that specific task or look into synthetic data.

Synthetic data should have all the required qualities of the real data. The only difference is that the synthetic data was generated by a simulation instead of gathered from the real world. Synthetic data has the potential to solve any kind of issues found in the process of gathering a dataset from real world. Generating synthetic data can be faster and easier than collecting a similar dataset the conventional way. Especially the labelling, which can

be the last and most time consuming part of building a dataset, is completely automated and accurate in synthetic data. Generating a perfect synthetic dataset may seem unrealistic in practice and sometimes that is the case. Models that are trained on synthetic data will probably perform accurate predictions on synthetic test data but the results can be entirely different when the solution is applied to real world instances. Zimmermann et al. (2018) got the highest performance in some of their experiments when they used a combination of synthetic and real data.

1.2 Big data for robotic manipulators

Applying ML to robotic applications comes with many challenges and problems to solve. It is quite safe to assume that the robotic task has something to do with controlling the robot. Otherwise, it is probably not strictly a robotic task. Controlling a robot accurately requires knowledge about the robot's structure and properties. Collecting movement data from a robot may include recording the joint angles and torques. Additional data can be collected with Inertial Measurement Units (IMUs) that can measure angular velocities and linear accelerations. Whatever the goal of the task might be, collecting a sufficient amount of data by moving the robot around and recording data is time consuming and expensive. Another downside is that the data is only descriptive for that specific robot model. Any changes in configuration or dimensions of the robot will cause the data to be no longer descriptive of the new robot configuration. This is not a huge problem if the goal is to apply the solution only to the specific robot model that the data was collected from. A better direction to take would be to find a solution that works for several different robots. Extending the data collection to many different kinds of robots would be even more time consuming and expensive and it would not even solve the problem. The dataset would be of no use when the solution is applied to a robot that did not participate in building the dataset. Building a dataset on measurements from every existing robot type is not feasible and even that dataset could not include robots that will be built in the future. A solution for this problem could be found by utilizing synthetic data together with measured robot data.

The approach of learning the structures of multiple robots works as long as the robots stay the same. A model trained from the data could learn the numbers associated with certain robot structures but that solution will stop working when the robot picks up a different load and the mass of the structure changes. The goal here is not to learn from many robots and hope that it will work later. Instead, the goal is to learn deeper features that are common to the robotic structures so that the solution could generalize to all robots regardless of mass or structure.

Recording a dataset with enough variety to generalize in any robotic structure is not feasible. Generating the data with multiple simulators is a suboptimal solution because someone has to make the simulators. This work explores a solution where the data is generated

by one reconfigurable simulator that can simulate any robotic structure. A simulator like that can simulate randomized configurations with randomized parameters.

To answer to these needs, we have developed our own reconfigurable multiple robot simulation model and used it to generate data. With the simulator, the quantity of the needed data is not an issue because it can generate more than enough data. Generating data by simulating does still take some time so the data will not be available instantly. The variety in the data is also not an issue because the randomized parameters make sure that there is not any kind of bias in the choice of configurations. Except, if the user specifically wants to have some bias in the parameters. This freedom of choice also opens the door for user error in the parametrization phase. The generated data will only be as good as the parameters.

1.3 Variation of system parameters and structures

Variation of system parameters and structures is important when the goal is to learn deep features from a system. For example, if the model is trained with data from a manipulator in which all links weigh 10 kg, the model will work fine for that specific set of manipulators. However, as soon as there is some change in the structure like a weight increase to 11 kg, the model performance decreases.

Changing payloads are the source of many challenges in manipulation tasks. Adding a payload to the end of a manipulator changes the required control forces to move the manipulator in a desired way. Being able to determine the weight of the payload quickly would be a great help in determining the optimal control parameters for a manipulator in every situation.

Data augmentation is a relatively common approach for increasing the size and variety of the dataset. The augmentation involves modifying the existing data in various ways to generate additional data. For example, a dataset of images can be augmented by rotating, zooming, mirroring, cropping and many other ways. These are fairly simple operations that can even be performed during the training of the network to save storage space (Géron 2017). Unfortunately, data augmentation is not that simple for time series based data from mechanical structures. Applying some predetermined effect on a time series data will probably corrupt most of the information in it. Generating synthetic data by simulation can be considered data augmentation but it is a more complicated process.

1.4 Inertial measurement units

Inertial measurement units (IMUs) are motion-sensing sensors. The sensors measure translation and rotation by utilizing gyroscopes and accelerometers. Some sensors may include magnetometers and barometers as well. The gyroscopes measure the rotation of

the sensor around three perpendicular axes and the accelerometers measure the acceleration along the same axes. The combination of these measurements produces detailed information about the movement of the sensor in a 3D space. IMUs are inexpensive and they generate highly descriptive data. For these reasons, they are used in many applications including smart phones, vehicles and robots.

Measurements can never be perfectly accurate and much like any other sensor type, IMUs suffer from multiple accuracy reducing factors. The sensors often have internal digital compensation for many types of errors but the sensor output will always have some degree of drift, random walk and random noise. These errors must be taken into account when the data is applied to an application.

1.5 Convolutional neural networks and time series data

Convolutional neural networks (CNN) are one of the most popular and powerful networks for machine learning problems. They are versatile because they can extract features from the data by themselves without guidance. Manually extracting features by feature engineering used to be the popular method. It involves analyzing the data, selecting the best features and generating descriptive values from the dataset such as averages, standard deviations and ratios of different features. Some of these engineered features could be very useful but others may not be helpful at all. CNNs find the useful features by themselves without the help of a data engineer. Even though CNNs are able to extract features, their ability to do so depends on the quality of the input data.

CNNs are considered black box models which means that the decision making process is hard or impossible to explain Géron (2016). The opposite of this is a white box model whose internal logic can be observed and understood. An example of a white box ML model is a decision tree. Decision trees simply ask series of questions about the data and each question directs the process towards new questions until it reaches one of the leaf nodes that states the output of the model. The logic is completely transparent and easy to understand.

CNNs processes a small amount of data points close to each other at a time. The outputs of the convolution operation depend the values of the data points and their locations in the data structure. CNNs only work when the data has a meaningful order. For example, the order of pixels in an image is equally important as their color values. Changing either of those will cause the image to lose most of its information. Time series describe the value of some measurement over time. Time series can be considered as one-dimensional images, which makes them a suitable data type for CNNs. However, a one-dimensional time series will most likely not include enough data for the application in question. Multiple time series can form a two-dimensional structure similar to images. However, only the time dimension has specific order unlike the order of the separate time series. Using two-dimensional convolutions on this kind of time series data is possible but the results

depend on the order of the data. Creating this kind of two-dimensional time series dataset can be considered feature engineering because the data engineer chooses the order of the data manually.

2. STATE-OF-THE-ART

This work is connected to several state-of-the-art researches from different aspects. This chapter presents some works on simulated and physical reconfigurable mechanisms due to the reconfigurable and modular structure of Aaria. System parameter identification is one of the tasks this work can help to solve. Human activity recognition has a different goal than this work but they share a significant amount of research methods. This work produces and utilizes synthetic data so this chapter also includes a section about works done on DL involving synthetic data.

2.1 Reconfigurable manipulators

Kereluk and Emami (2015) present a reconfigurable manipulator platform they call MARS. The manipulator is built out of six modular blocks, which each have one prismatic joint and two revolute joints. The joints are mutually perpendicular. The whole structure has 18 degrees of freedom (DOF). The reconfiguration is done by locking the movement of some of the joints so that the structure emulates a manipulator with fewer DOFs. The joint modules have similar structure but the size is different. The modules are bigger near the base and the size decreases towards the tip of the manipulator. The weight is a challenge for the manipulator because the modules are heavier than conventional robot joints. The heavy structure also imposes some requirements on the joint motors so they had to reduce the weight of the structure and choose motors with enough torque. The configuration of the manipulator is designed and simulated in a software. The software can perform various types of simulations of the manipulator and provide information about the configuration. Gathering data from the movements is also possible. They tested two different manipulator configuration on the same task to demonstrate how the MARS manipulator expresses the advantages and disadvantages of both manipulator types. The tests reported clear differences between the accuracies of the two configurations for the same task. There were also differences in power consumption between different configurations.

Hong et al. (2016) presented a concept of a new joint module that they used to build an anthropomorphic arm manipulator. The modules are revolute joints that have two perpendicular input connectors and one output connector that is aligned with one of the input ports. The modules can be manually assembled into a desired configuration by using special link modules between the joints or connecting the joints directly to each other.

Viegas et al. (2017) presented a reconfigurable grid-based parallel manipulator. The limbs of the parallel manipulator are connected to mobile robots that move along a rail grid to control the manipulator. The grid structure allows the manipulator to have larger workspace compared to a conventional parallel manipulator. In addition, the manipulator can

be reconfigured by moving the mobile robots from one rail to another. This design allows the manipulator to have flexible workspace shape and size and the possibility to have multiple manipulators on the same rails.

Do et al. (2016) developed a simulator for modular and reconfigurable robots. The robot models were built by selecting the needed modules from the user interface. The simulator can automatically generate codes for kinematics and controllers.

Zhang et al. (2018) created an underactuated self-reconfigurable robot. The robot consists of active and passive modules that can be connected to each other via a lock-key docking system. The modules can be docked in passive mode so that the created joint moves freely or in active mode where the active module's servomotor controls the joint movement. A reconfiguration algorithm that plans and schedules the steps needed to achieve the new configuration was also proposed. A scenario was demonstrated where a humanoid-like configuration disconnects an "arm" module and uses it and the other arm to replace "leg" modules. The robot was controlled by an external centralized offline controller.

Zhang et al. (2016) presented a modular self-reconfigurable robot. Each module has a hexagonal shape, three omnidirectional wheels and a docking mechanism that is connected to the module by a pitch joint. The omnidirectional wheels allow the module to rotate in place and move to any direction. The docking mechanism can attach to any side of another module excluding top, bottom and the side with the active docking mechanism. Two docked modules can move by using the wheels or by using the pitch joint to perform a crawling movement. On flat surfaces, the wheels were reported to be the faster movement option but on uneven surfaces, the crawling proved to be faster.

Jing et al. (2018) created a high-level task planning system for modular robots. They created a simulation environment where they can design and test new robot configurations. The configurations were compiled into a library along with the properties and capabilities. The high-level mission planner would then reference the library to find configurations that are capable of performing the needed tasks and fit the restrictions so that they are able to complete the given task. The library based solution is faster and more reliable in the mission planning phase than generating a new configuration on the spot. However, building up the configuration library is slow manual process that has to be done beforehand. The library was populated with the help of volunteering students. The system is designed for cube-shaped SMORES-EP (Davey et al. 2012) robot modules. Each module can dock to other modules by using any of its 4 magnetic faces. Two of the faces on the opposite sides of the module can be used as wheels to drive the module around. One face can tilt 180 degrees as well as rotate. The last face does not move but the edges can slide along the surface when the wheels are driving the module forward. The modules offer quite flexible reconfigurability but some of the limiting factors are the maximum torques of the motors and the strength of the magnetic connection between modules.

2.2 Identification of system parameters

Bjurgert et al. (2017) applied Adaptive Boosting algorithm to identify system parameters. Adaptive Boosting is a machine-learning algorithm that iteratively updates the model focusing on the samples that were misclassified. They demonstrated their findings by generating data with an example function and used that data to fit the Adaptive Boosting model. After a few iterations, their model fit the example function. The boosting method reached similar performance to the well-known prediction error method.

Wang et al. (2018) discovered a new experimental method to identify system parameters. The goal was to reduce the vibration of a truck driver and the seat. They modelled the system as a 5 DOF lumped mass-spring-dashpot system and parametrized it by values taken from relevant literature. Next, they recorded vibration data from several places on a real truck seat and driver while the truck was running. The system's modal resonant frequencies were calculated from the measured data and the modelled system's parameters were tuned so that they would share the same modal resonant frequencies. After identifying the system parameters, they implemented vibration-dampening system with optimized PID controllers.

Gao et al. (2018) proposed a parameter identification method for improving the accuracy of 6 DOF industrial robots. The inaccuracies in the robot are mostly due to the difference between the nominal DH-parameters and the actual parameters of the robot. They use least squares method to calculate the actual structure parameter vector, which they use to compensate the inaccuracy of the nominal structure parameters.

Ahandani et al. (2018) proposed two estimation methods for improving the performance of backtracking search optimization algorithm in identifying parameters in chaotic systems. The first method applies a trial population for the algorithm that speeds up the convergence and deepens the search. The second method groups the population to enhance the exploration of the search space. They tested the methods on 10 different chaotic systems and found that both methods improved the performance of backtracking search optimization algorithm.

Čepon et al. (2018) presented a method for improving the accuracy of joint-parameter identification for multibody-systems. The method uses an algorithm that finds the optimal equilibrium for the system so that the different parts of the structure affect each other as little as possible. They demonstrate the method by estimating the joint stiffness parameters for a three-link system. The structure was vibrated in a neutral horizontal configuration and in the calculated optimal configuration. The joint stiffness was estimated based on the frequency response of the system. The estimations made with the responses from the optimal configuration were more accurate than the estimations from the neutral configuration.

Jung et al (2018) proposed a method for identifying dynamic parameters of robot manipulators. The method involves measuring simulated movement and torque from a preset trajectory and comparing the values to calculated values. The presented method progresses sequentially from the tip of the manipulator to the base. They found that the sequential approach was significantly faster than a simultaneous one. However, even the faster sequential method took 51 minutes to identify the parameters of a 3-DOF manipulator.

2.3 Human activity recognition

Human activity recognition has become popular in recent years with the help of machine learning and Inertial Measurement Units (IMUs). Often these studies involve several test subjects performing tasks while wearing IMUs on their body. The recorded IMU data is then processed in various ways to extract the desired information. Sometimes it is enough to measure the movement with smartphone's internal sensors while it is in the test subject's pocket. Simple human activity recognition tasks deal with trying to distinguish simple activities like walking and sitting (Kasnesis et al. 2018). There are also numerous medical applications where sensors and machine learning is being used to for example in prediction of Parkinson's disease (Caramia et al. 2018).

Caramia et al. (2018) explored the possibilities of classifying Parkinson's disease from gait. They organized a test where they strapped IMU sensors to patients with different stages of Parkinson's disease and gathered data from their manner of walking. They also gathered similar data from a control group of healthy age-matched people. The sensors were placed to various parts of the body. They processed the data and extracted features like the ranges of motion of the joints and spatio-temporal features like step length and duration among others. The features were then split into different groups by various criteria to find out which features contribute to the classification accuracy the most. They tried several ML techniques and a voting classifier combined from the other classifiers. The classification accuracies of the simple ML techniques reached accuracies between 63 % and 80 % and the voting classifier was able to reach 96 % accuracy. The ranges of motion of hips, knees and ankles proved to be the best features for classification.

Zimmermann et al. (2018) presented their solution for a task where the goal is to assign IMU sensors to correct segments of a biomechanical model of a human lower body and align the sensors within the segments. The experiments used both recorded real IMU data and simulated IMU data. They achieved 94 % accuracy on the assignment problem when they used only the real IMU data. However, mixing simulated IMU data into the training set decreased the accuracy to around 92 %. Training with only simulated data and testing with real data only reached 68 % accuracy in the assignment task. In the alignment task, they found that combining real and simulated IMU data significantly reduced the alignment error. Both tasks also benefited when they used a model that was pre-trained on simulated IMU data.

There are multiple datasets freely available for various things including IMU data for human activity recognition. One of these datasets is OPPORTUNITY (Roggen et al. 2010). An article by Ordóñez et al. (2016) explores the use of deep convolutional and recurrent neural networks to recognize human activity. The combination of CNN and LSTM networks works well on human activity recognition because first, the CNN part can fuse, detect and extract useful features from different sensors and then, the LSTM part can learn the temporal representations of the features. They refer to their network architectures as DeepConvLSTM. They tested the architecture on the OPPORTUNITY dataset, which contains recordings from different sensors attached to human body and different objects while the person performs everyday activities. They compared their architecture to a baseline CNN architecture and their network performed better in all their tests. They found that their architecture is significantly better at distinguishing similar activities such as opening and closing a door. This advantage is due to the LSTM layers' ability to process the order of the features.

Ranao and Cho (2016) performed human activity classification by using data from smartphones with convolutional neural networks. The smartphones recorded acceleration and angular velocity data in the test subjects' pockets while they performed six different activities. They split the data into small segments with some overlap, regularized it and fed it into the CNN. They tried several network architectures and hyperparameters. The final classification accuracy was about 95 %.

Kasnesis et al. (2018) present their approach to human activity recognition by using CNN and late sensor fusion. By late sensor fusion, they mean processing different sensor signals together at a later stage in the network instead of doing it at the beginning or even in a preprocessing stage. The architecture of their PerceptionNet has two 1D convolutional layers with ReLU activation functions followed by 1D Max Pooling layers. Pooling layers are followed by dropout layers. The next layer is a 2D convolutional layer, which is followed by unconventional Global Average Pooling layer. The rest of the network consists of a dense layer with Softmax activation function. They trained and tested the network on UCL and PAMAP2 datasets. The UCL dataset contains acceleration and angular velocity data recorded by a smartphone while the test subject performs six simple actions. The PAMAP2 dataset contains similar data recorded from three different body parts while the test subjects were performing 12 different activities. They found out that applying 2D convolution to the third layer gave better results compared to first and second layer. They also tested different options for the last hidden layer and found that Global Average Pooling outperforms Max Pooling, Global Max Pooling and a 250-neuron dense layer. PerceptionNet managed to get 97.25 % accuracy on UCL and 88.56 % on PAMAP2 at best. These accuracies beat the previous records by over 3 %.

Hsu et al. (2018) developed a wearable sensor network for recognizing 10 daily and 11 sport activities. The IMU sensors were placed on the wrist and ankle of the test subjects. They use a multi-step preprocess to obtain high quality data. Possibly useful features are

extracted from the data by feature engineering. The extracted features were then normalized and reduced by principal component analysis. The activity recognition was done with a least squares support vector machine. The final classification accuracies for daily and sports activities were 98.23 % and 99.55 % respectively.

Margarito et al. (2016) investigated sports activity recognition by template matching using only one accelerometer. They recorded data from several different test subjects performing the sports activities while wearing the sensor on their wrists. Some of the data was used to generate templates that express periodical features specific to each activity. The classification was done by matching the test data with the most similar template. The template matching method did not perform as well as some other tested methods such as logistic regression and artificial neural networks. However, the template matching method maintained its performance better than the better performing methods when it was used on an unseen dataset.

2.4 Synthetic data

Horn and Houben (2018) demonstrated how they have used a game engine to gather synthetic data for machine learning purposes. They simulated a parking lot and gathered data about available and occupied parking spaces. The benefit of generating simulated data this way is that the simulator can label the data automatically. The labelling process for real video data would be manual and very time-consuming work. The proposed method seems promising as the model trained with the synthetic data reached same performance as a model trained with conventional video data.

Hoffmann et al. (2019) studied the formation of creases in crumpled sheets of paper. The goal was to be able to predict the locations of ridges on the paper using data about the valleys in the paper. Gathering enough experimental data for the task proved to be too slow and expensive. The solution for this problem was to generate data by simulating the folding of paper and collecting synthetic data. The simulated data could not achieve the same complexity as a physical crumpled sheet of paper but that was not an issue. They augmented the experimental data with the synthetic data and managed to get better results than before.

Hanel et al. (2018) trained traffic sign classifiers with multiple synthetic datasets. The synthetic datasets are created by using a game engine to render realistic scenes in traffic setting. The advantage of synthetic traffic sign data is that the signs can be automatically labelled with pixel perfect accuracy without any manual labor. In addition, the weather and some other external conditions can be easily modified to increase the data variety. On the other hand, the simulated environment is not photo-realistic, which may affect the classifier performance. Additionally, different datasets may label the backsides of the signs and the poles as part of the traffic sign or as a different object. They trained the classifiers with synthetic data and tested it with real life data to evaluate the feasibility of

using synthetic data for a real life application. Despite the challenges, they reached overall accuracy of 98.7 % and showed that synthetic data can be used to train a well performing classifier.

Yue et al. (2018) presented a method for generating synthetic data for machine learning applications. The method uses some existing data and k-nearest neighbor model to obtain a probability distribution from the data. Then they generate random new synthetic data according to the obtained probability distribution. They tested the synthetic data by training some models with it and comparing the results to models trained with real data. The achieved performances were close to each other proving the method functional. The method works for data augmenting purposes but the required initial data disqualifies it from applications where no data is available.

Malmgren-Hansen et al. (2017) did research on improving the target recognition accuracy of a synthetic aperture radar by using synthetic training data. The available data for the task was limited so they created synthetic data by simulating the reflections and scattering of the radar signal on a 3D model and the background. They use the synthetic data to pretrain the CNN before training it with the real data. The pretraining increased the accuracy made the network converge faster, especially when using a small dataset.

Sobie et al. (2018) generated data for roller bearing fault detection task by simulating the bearings. They trained multiple models on the synthetic data and tested the models with experimental data. The models trained on synthetic data performed better than the models trained on experimental data. The findings indicate that in this task, increasing data variety is more important than increasing data quantity.

Zhang et al. (2019) proposed a method for generating synthetic training data for thermal infrared tracking applications. Labelled infrared training data is not as readily available as RGB training data and the features of these data types differ significantly. Their solution to this problem was to use translation models to convert RGB images into thermal infrared images. They used a conditional adversarial network for the image conversion. The converted data was then used to train three different networks to find the best features for the tracker. They also added handcrafted motion features to the data to further improve the performance of the tracker. The resulting model performed better than the state-of-the-art models that used primarily handcrafted features.

Sakaridis et al. (2018) presented a method for semantic segmentation in foggy images by using synthetic foggy images. The synthetic foggy images were created by referencing a depth map, semantics and the clear image to accurately model the edges in the pictures before applying simulated fog. Their segmentation method works by training the model with increasingly foggy synthetic images and adapting the model to less foggy real images. The model improves gradually and finally it is able to perform semantic segmentation for real images with dense fog.

Rad et al. (2018) used synthetic data for predicting 3D object poses in images. Their method involves a combination of three networks that learn to map features from real images to synthetic images by overlaying a synthetic 3D model on top of the real object in matching pose. This method enables them to train the networks with a combination or real and synthetic images. The synthetic dataset was generated by sampling the 3D models in a random pose, rotation and scale. The models were then placed on random backgrounds. The final model had better performance than the state-of-the-art models in 3D pose estimation for objects and hands.

Marcu et al. (2019) created a synthetic 3D dataset of built environments to detect safe landing areas for drones. They used Google Earth to gather aerial pictures and depth maps of landscapes with buildings. The pictures were then annotated with surface types: horizontal, vertical and other. They trained two CNNs on the synthetic data and the larger one reached 84 % classification accuracy based on only image inputs.

3. AARIA

I have built a modular and reconfigurable simulation platform for robotic manipulators and named it Aaria (Hautakoski et al. 2018). Aaria has a modular structure that allows it to reconfigure its structure based on Denavit-Hartenberg (DH) parameters. This makes the model flexible and enables it to be used in various applications. The primary purpose of Aaria is to generate synthetic IMU sensor data for ML applications but it can also be used for secondary applications such as visualizing robotic structures and testing control methods.

Aaria is a model built in Matlab Simulink environment by using Simscape Multibody library. Aaria can simulate a wide variety of manipulators with open chain structure and up to six joints. The joints can be revolute or prismatic joints. The simulated manipulator is mounted on a floating platform. Each link of the simulated manipulator has two simulated triaxial IMUs modelled according to the specifications of Analog Devices tactical grade ADIS16485 sensor (Analog Devices, 2018). Aaria was built to be modular and most importantly to be simulated in parallel with randomized parameters.

3.1 Denavit-Hartenberg parameters

DH parameters are a set of parameters used to describe manipulator structures introduced by (Denavit & Hartenberg 1955). Aaria uses modified DH parameters introduced by (Craig 1986) where the coordinate frame placement and order of operations is further optimized from in the original notation. The modified DH parameters of a manipulator structure are determined by first placing a coordinate frame on each joint so that the z-axis is aligned with the rotation axis of the revolute joint or the movement axis of a prismatic joint, x-axis points towards the next joint and the y-axis completes the coordinate frame without any specific purpose. When moving from one coordinate frame to the next, the amount of twist in the link around the x-axis is described by parameter α. The length of the link along the x-axis is described by parameter a or r depending on the source. The twist around the z-axis is described by parameter θ. Finally, the length of the link along z-axis is described by parameter d. In revolute joints the twist around the z-axis is the rotation angle of the joint and the value may not be given because it varies as the joint moves. This applies similarly to prismatic joints and the d parameter. The coordinate frame assignment in modified DH notation is illustrated in Figure 1

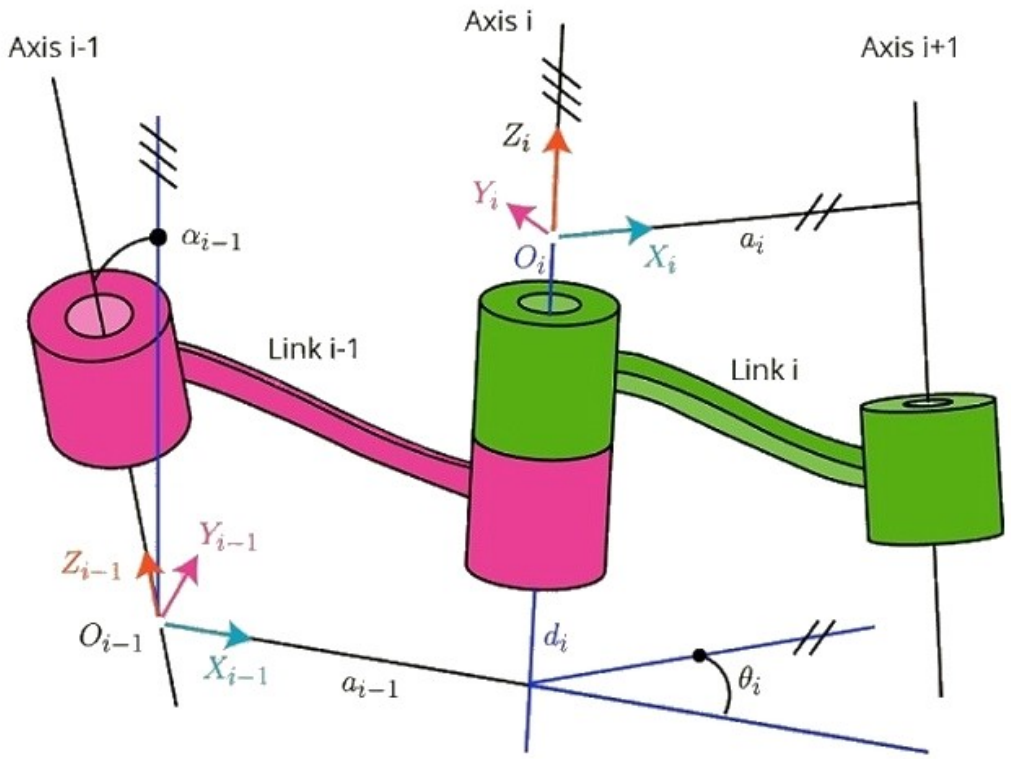

Figure 1. Coordinate frame assignment according to modified DH convention. Figure adapted from (Puri 2017).

DH parameters are commonly used to describe manipulator structures. The combination of parameters and the rules of using them offers a precise way of defining a structure while using very few amount of parameters. The parameters are non-unique but they can describe the dimensions of a manipulator precisely. However, the parameters cannot describe the manipulator pose consistently without additional information. The pose depends on the joint rotation angles that depend on the zero angle of each joint. The manipulator manufacturers can define the zero angle arbitrarily so there is no universal way to define pose with DH parameters. Usually the DH parameters are only used to describe the structure and the pose is defined in other ways. Even though the parameters are versatile, they are unable to perform a rotation around the *y*-axis in a single link. However, this flaw can be worked around by adding a fixed joint to the parameters. Overall, DH parameters are well suited for describing manipulator structures so building a simulator based on them is a reasonable solution.

The DH parameters focus specifically on the four parameters defining the properties of each link. This information is not enough for defining an entire manipulator since it lacks the information about joint type. We treat joint type as a fifth parameter so we can store all the relevant structural information in the same table. The joint type parameter can be an integer from 0 to 5 and each value is mapped to a different joint type. The available joint types in Aaria are empty, motion controlled revolute, motion controlled prismatic, torque controlled revolute and force controlled prismatic. The joint type parameter defines the control mode in addition to the joint type.

3.2 Reconfigurable and modular structure

The goal with Aaria was to be able to simulate manipulators that have randomized structures. The randomization of the link length of the manipulator is just a simple number modification but changing the number of joints or joint types is a more complex operation. This functionality was implemented by using variant subsystems. A subsystem is a group of blocks combined into one system that has a number of input and outputs. A variant subsystem is like subsystem but instead of one, it has multiple systems inside of it. Only one system inside the variant subsystem can be active at a time. The active system variant is determined by the joint type parameter given to the variant subsystem. The manipulator structure is built out of six identical modular variant subsystem blocks. Each variant subsystem has five different variants inside it. The variants are motion controlled revolute joint, motion controlled prismatic joint, torque controlled revolute joint, force controlled prismatic joint and an empty joint. Any joint in the six-part chain can be set to be any of the variants. The purpose of the empty joint is to shorten the kinematic chain so that the model can simulate manipulators with less than six joints. The modular structure is illustrated in Figure 2 and an example of a resulting manipulator is presented in Figure 3. The overview of the model structure is presented in appendix A.

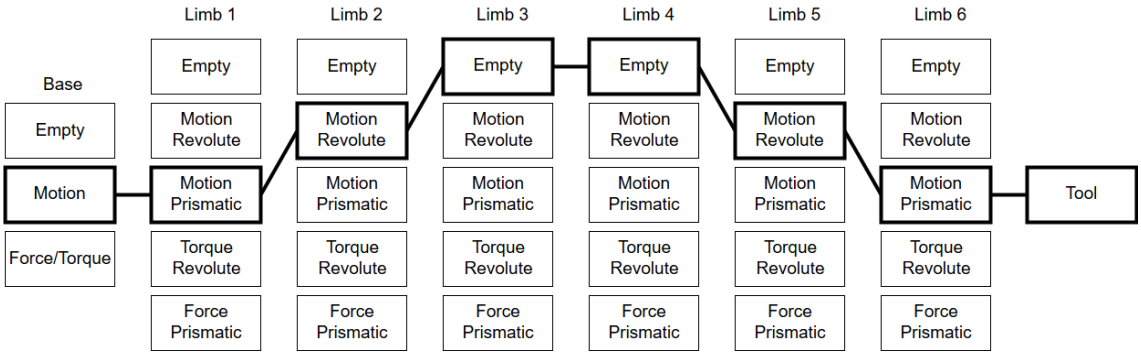

Figure 2. The configuration is built by selecting one joint type from each Aaria module

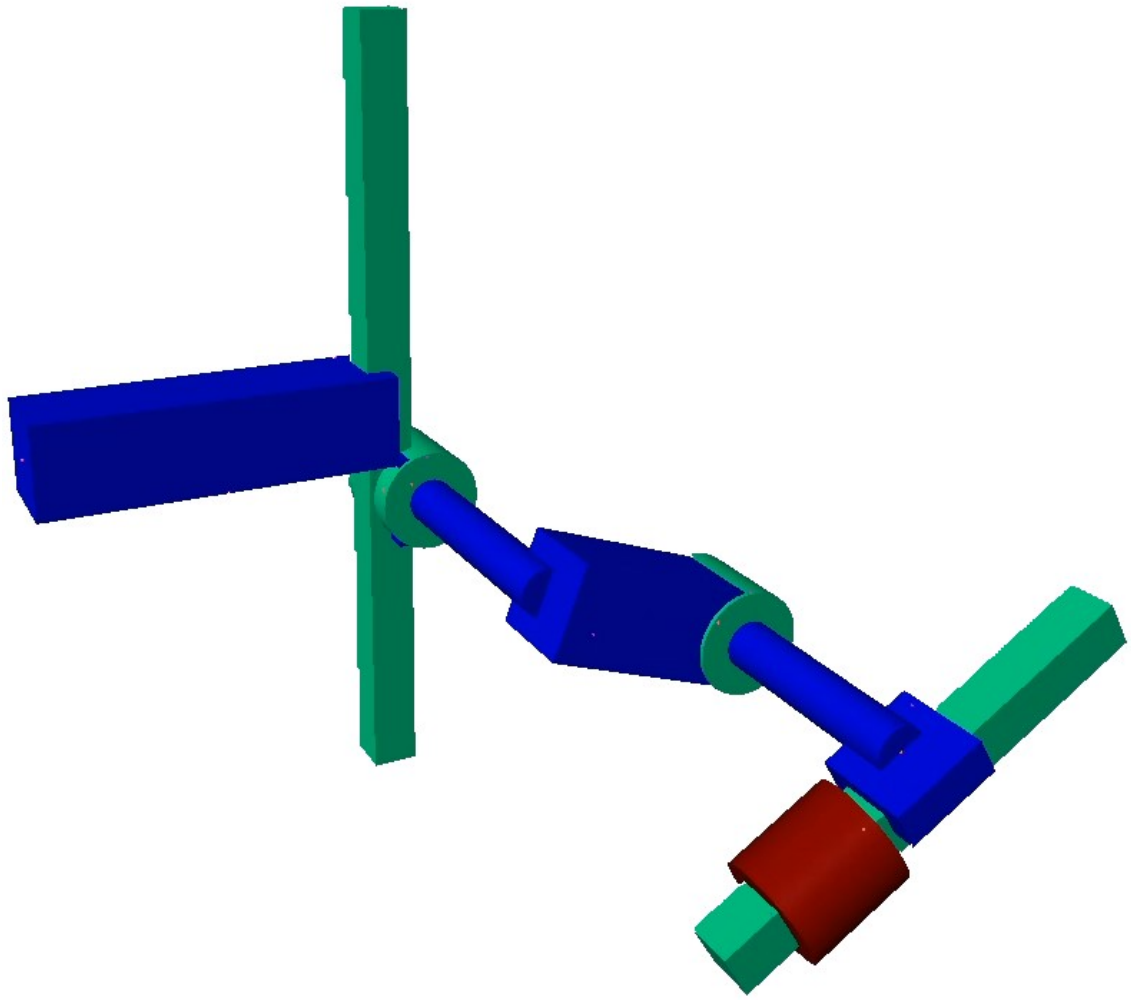

Figure 3. A possible manipulator configuration based on the joint type parameters defined in Figure 2.

Aaria was built to follow the modified DH-parameter notation introduced by (Craig, 1986). Building a structure from modified DH-parameters is done easily in a conceptual level by rotating the coordinate frame by α radians, placing a link with a length of a, placing a revolute joint and then placing another link with a length of d. However there are some additional steps in the process such as transforming between Simulink and DH-coordinate frames between different steps to make rotations around the correct axis and to place the links pointing to the right direction. Placing a link in the model first requires a translation to the middle of the link where the solid representing the link itself is placed and then another translation to reach the end of the link solid. Prismatic joint subsystem is very similar with the revolute joint subsystem. The difference between these subsystems is that the revolute joint is replaced by a rigid transform block that is rotated by θ radians and the last link is replaced by prismatic joint.

Each module includes a sensor module, which places four simulated IMU sensor models on the link. The number of sensors used be just two but we doubled the amount of sensors

to gather data with more detail. The most important part of the sensor module is a Transform Sensor block from Simscape Multibody. The block senses the relationship of the two frames connected to it and outputs the results. The outputs can be configured to include a wide variety of different ways to measure the 3D relationship of two frames but in this model we chose angular velocity around x, y and z-axis and linear accelerations along the same axes. The sensor module is attached to the beginning of the link so each sensor in the module has to have a unique offset that places it on the link. The sensors are placed on the link based on the relative distance from the beginning of the link. The sensor locations are manually set to 10 %, 40 %, 60 % and 90 % of the link length but the placement can also be randomized. The sensors are placed on different edges of the link. Each sensor has a randomized misalignment that rotates it so that the sensors do not have a mutual coordinate axis. The Transform Sensor block does not measure gravity so it has to be manually added to the measurement. This is done by measuring the rotation of the sensor and multiplying it with the gravity vector and summing it to the linear acceleration output bus. The final part of the sensor module are the noise blocks. Each signal has three kinds of noises: in-run bias that slowly increases the measurement value over time, normally distributed random noise and a running sum of uniformly distributed random numbers to model random walk of the sensor. The noise model of an IMU gyroscope is presented in Figure 4.

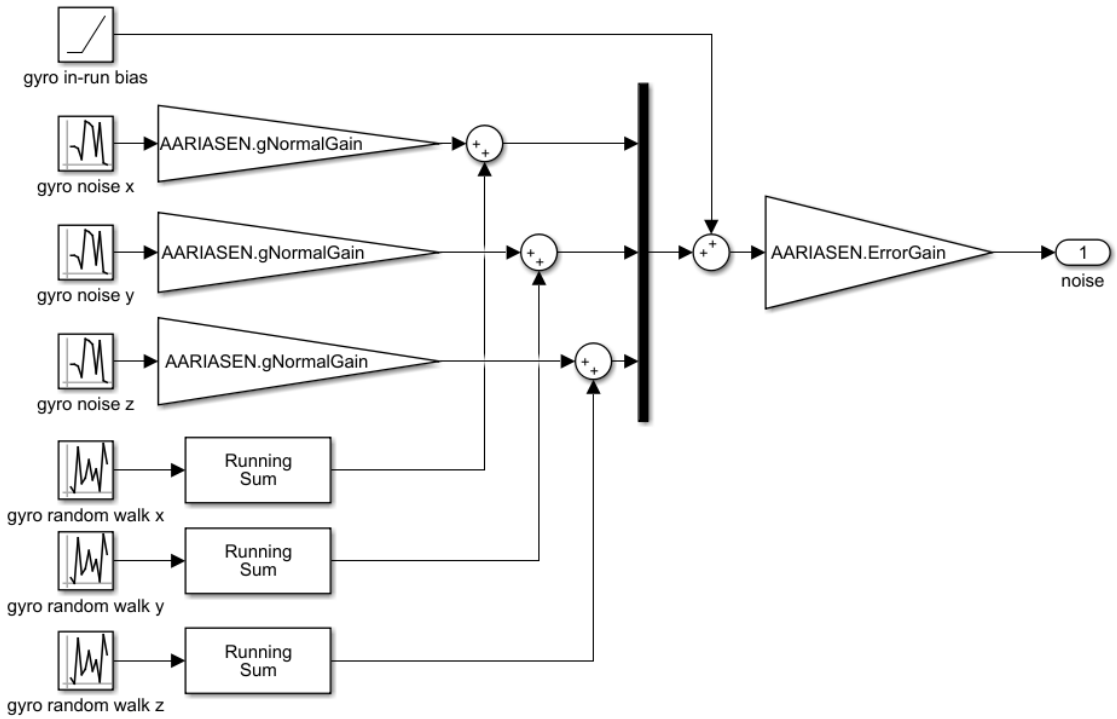

Figure 4. Noise model of a gyroscope in an IMU model adds in-run bias, normally distributed random noise and random walk the angular velocity signal.

3.3 Simulation outputs

The output of the model includes time series of measured joint angles and the outputs of IMU sensor from each link. The model also saves the manipulator configuration parameters and environmental parameters such as gravity vector and randomization parameters. The joint angles and IMU outputs are saved so that they can be used as training data for our machine learning models. The configuration parameters like the DH parameters and the joint types are saved so they can be used as labels for the measurements. Saving the configuration parameters allows us the recreate any past simulation. This is a useful feature to have for monitoring purposes because the visualization is normally disabled during simulations to speed up the process. The seeds for creating the sensor noise are also saved from every simulation. Not only does this allow us to recreate the exact noise from a past simulation but it also gives us the possibility of removing or altering the noise in a time series without a complete re-simulation.

The simulation sample time has a substantial impact on the simulation outputs. The sample time determines how often a value is sampled from the simulation. A short enough sample time is required for accurate capture of fast dynamics. Too long sample time results in undersampling of the signal and an oscillating signal might appear to have lower frequency than it does. This phenomenon is referred as aliasing. On the other hand, excessively short sample time slows the simulation down and increases the amount of recorded data, which increases the requirements for data storage and processing capacity down the line.

The randomized parameters create diversity in the simulated structures. The number of possible manipulator configurations depends on the resolution of the input parameters. Even with a small selection of parameter values, the number of configurations is very high due to the multidimensional nature of the configurations. For example the number of unique configurations will be over 600 000 with just 10 different values for link length and offset, 4 different values for rotation and 2 choices for joint type in a simple 2-link structure. The number of configurations grows rapidly as we increase the amount of links and the parameter resolution. Trying to achieve a full coverage of a manipulator configuration space with six links is unfeasible. Fortunately, a full coverage is not necessary nor even desirable. The ML algorithms do not need every possible data point to train a good model so a sparse coverage of the configuration space is good enough.

The simulation parameters are randomized by a function that produces a uniform distribution of random values. The randomization function is parametrized by the lower and upper bounds of the wanted output value range and the desired resolution of the output values. An example output distribution of the randomization function is presented in Figure 5when the target range is from 0 m to 1 m and the resolution is 0.1 m.

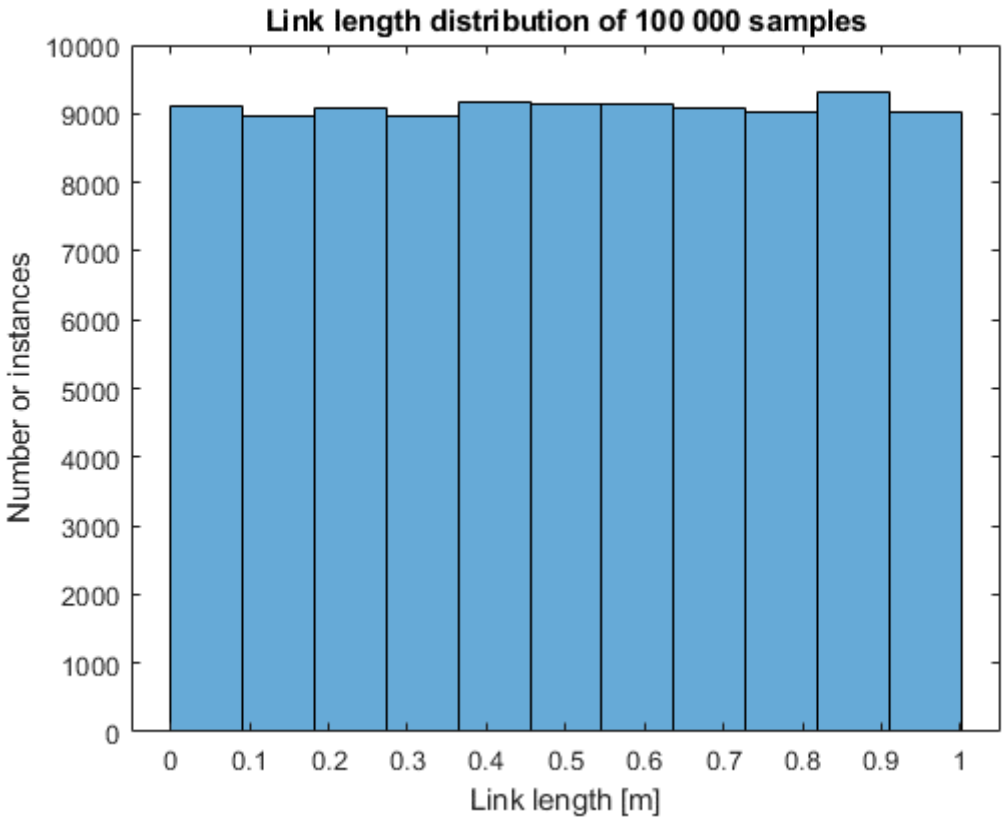

Figure 5. The randomization function produces a uniform distribution of random values in a given range with a given resolution

Each parameter is randomized individually and as a result, the parameters do not depend on each other. This causes the overall features of the manipulators to follow a normal distribution. This is illustrated in Figure 6 with a histogram of the sum of the first two link lengths. Both links have a uniformly distributed length but the sum of the link lengths is normally distributed.

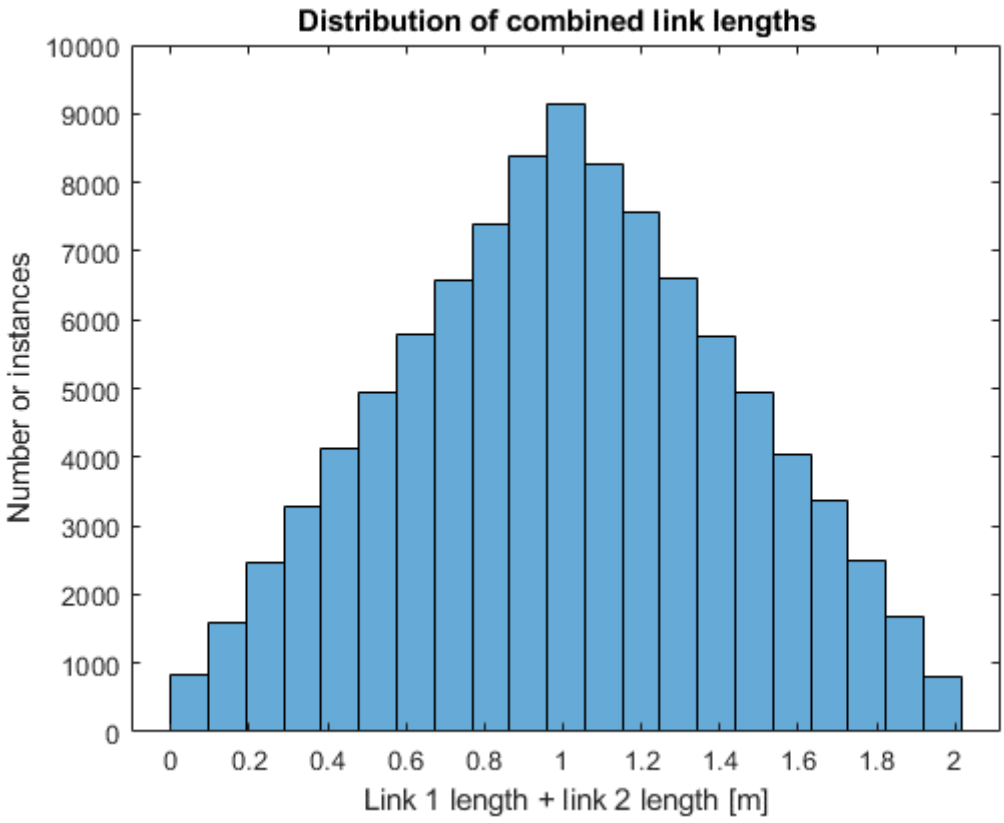

Figure 6. The total length of the manipulator follows a normal distribution.

Figure 7 presents the occurrence frequency of manipulator maximum reach based on the number of links in the manipulator. The development of the curves demonstrates how the coverage of the configuration space grows sparser near the extremities as the number of parameters increases. With 100 000 samples, the 2-link configuration is the only configuration that covers the entire space. The manipulator mass and center or gravity follow a similar distribution.

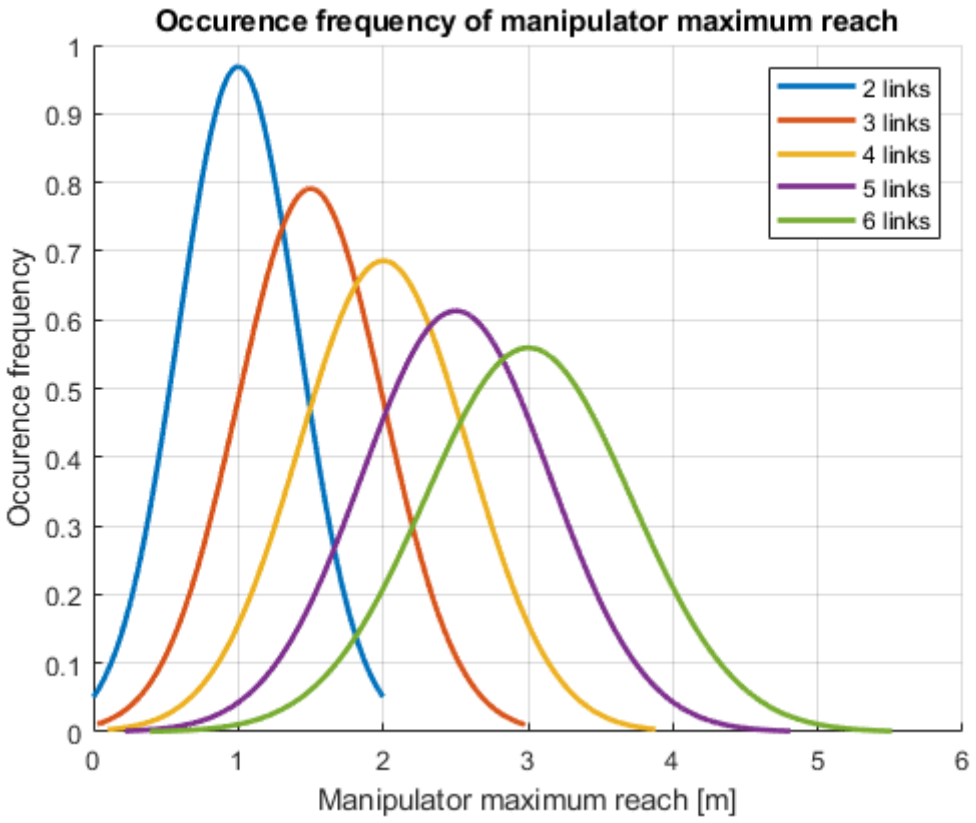

Figure 7. The maximum reach of a manipulator follows a normal distribution and the shape of the curve depends on the number of links in the manipulator.

3.4 Simscape Multibody

Simscape Multibody is a Simulink library of blocks that can be used to build and simulate mechanical 3D systems. The blocks include joints, bodies, constraints, force elements and sensors (Mathworks 2018). The blocks allow the user to build many kinds of mechanical structures with a variety of different dynamic relationships. Simscape Multibody is suitable for this application because it makes it easy to build and visualize 3D structures. The simulation can be viewed in real time or as fast as the simulation runs. A completed simulation can be played back at any speed and viewed from any angle.

Figure 8 illustrates the connection between the Simulink model, manipulator visualization and the transform matrices for a single revolute joint module. The transform operations are performed in order according to the modified DH parameter convention. The first operation is the twisting of the link from the previous module to the current one. In the example, the twist is $\frac{\pi}{2}$ radians around the x-axis and the transformation is presented in transformation matrix R1. The next operation is translation along the x-axis. The translation is presented in transformation matrix T1. The next operation is the joint rotation. The joint rotates around the z-axis according to the control signal. Transformation matrix R2

defines this rotation. The final operation is the link offset translation along the *z*-axis. The offset translation is presented in transformation matrix T2.

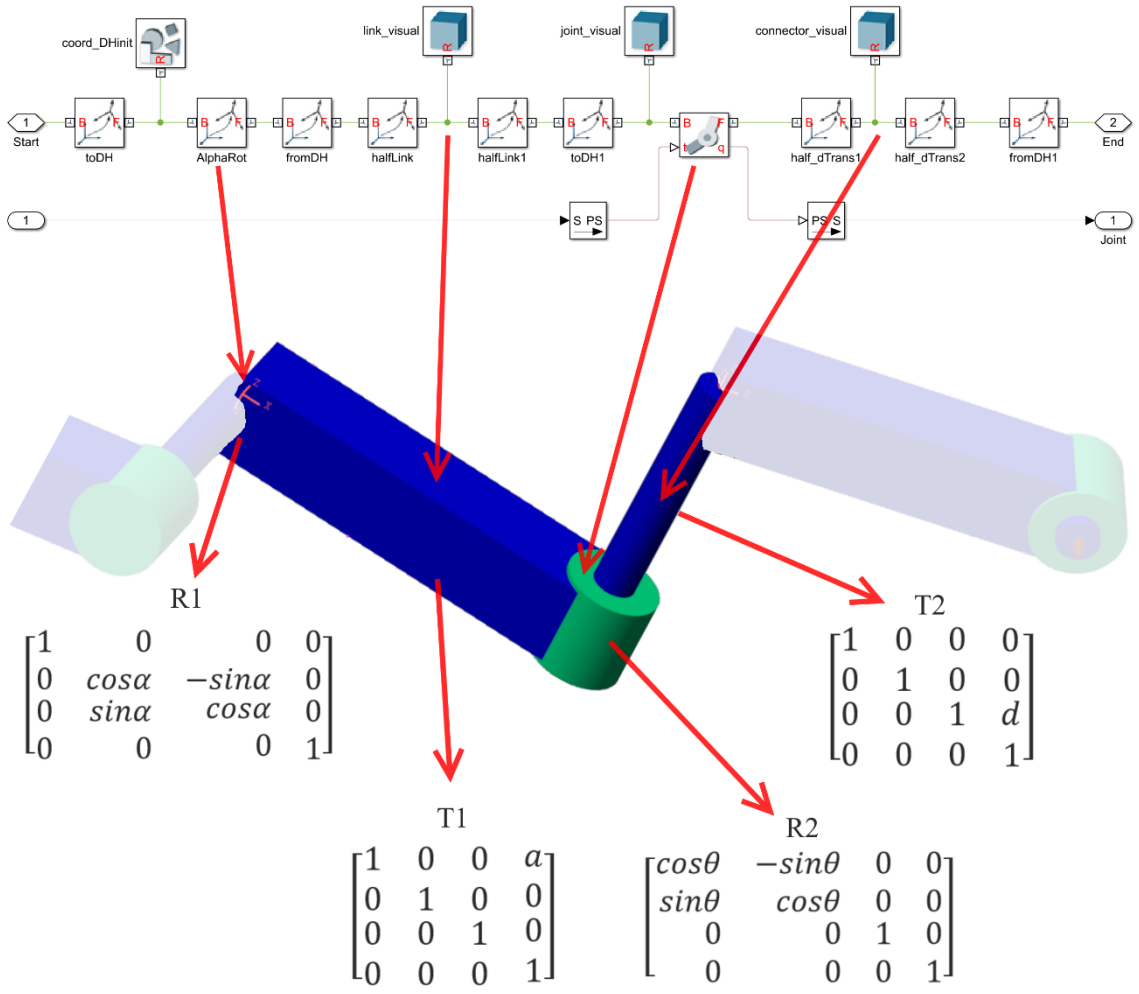

Figure 8. An example of the connections between the Simulink model, manipulator visualization and the corresponding transform matrices.

The Simulink model includes more transformation operations than specified in the DH parameters. This is because Simulink operates in a different coordinate system than the DH parameters. Every translation is also performed in two parts due to how solid objects are defined in Simulink. Figure 8 presents a simplified version with only the essential transformations

3.5 Simulation of position and torque control modes

The joints can be controlled with position or torque / force inputs. The position inputs make it easy to control the manipulator in a precise manner regardless of the physical properties of the manipulator. However, the way Simulink handles position-controlled

movement is not realistic. The position controlled joints can have unlimited torque range and they do not move through space in an intuitive way. They disappear from the previous location and appear in the next location. Position controlled joints do not react to any other movement in the structure and torque controlled joints do not react to movement caused by position controlled movement. To illustrate this surprising interaction, imagine the classical inverted pendulum structure where a pendulum is attached to a cart that moves on a rail. The goal is to swing the pendulum up and balance it upright by just moving the cart. In a case where the pendulum is attached by a freely rotating joint and the cart is position-controlled, the pendulum will always point straight down regardless of how rapidly the cart moves back and forth on the rail. The whole structure should be controlled by torque and force inputs to achieve the most realistic behavior. Despite the shortcomings of the position controlled movement, it is still suitable for generating measureable movement for the simulated IMU sensors.

Controlling the model with torques and forces is required for simulating accurate dynamical properties. Unfortunately, this control mode is difficult to configure so that it could control randomized structures in a reasonable manner to collect high quality data. Some of the options for configuring the torque and force control modes are the use of predefined torque and force inputs or implementing controllers to handle the trajectory tracking. Both methods require information about gravity and the manipulator configuration to generate a control signals that produce the desired trajectories. Implementing a controller in every joint is also computationally more expensive than using a predefined control signal. This means that the torque and force control mode is mostly useful when the control signals are manually tuned for a specific manipulator configuration. The position controlled simulations could be used to collect torque and force data but the simulations lack some dynamical interactions and the measured torque and force values tend to be very high and spiky since the motion control is not limited by a maximum torque or force.

3.6 Manual control and teleoperation

Even though Aaria was designed to be randomized in many different ways, there are possibilities for manual operation. Almost any parameter can be randomized to any value in a custom range, but just as easily those values can be set to constant. This allows the user to manually set the DH-table so that Aaria replicates the features of any manipulator the user wants.

There is also a version of Aaria that is joystick controlled. A USB-joystick can be used to control the joints in real time. This implementation was designed for a joystick with 4 analogue axes and 12 buttons. The analogue axes are used to control the first joints and two pairs of buttons are used to control the last two joints. Additionally there is button for opening and closing a gripper mounted to the end of the manipulator. Joystick input block receives the user's inputs and passes them forward. Dead zones are applied to the analogue inputs and half of the button inputs are turned to negative. The control signals are

then integrated and fed into the joints as position control. Joint limits are implemented by using integrator saturation limits.

The model records all joystick inputs during the simulation. A previous manually controlled simulation can be simulated again using the recorded joystick inputs. The recorded joystick inputs can be used as "learning by demonstration" test cases. However, collecting this kind of data involves a significant amount of manual work. Running the simulation in real time generates data much slower than during parallel offline simulations.

3.7 Visual Feedback

The simulated structure can be viewed in Mechanics explorer of Matlab. Rigid structures are coloured blue and joints are coloured green. Links representing DH-parameter *a* are shaped like rectangular rods and links representing DH-parameter *d* are shaped like cylinders. Revolute joints are green cylinders and prismatic joints are green rectangular rods. The tool is a red cylinder. It is also possible to draw coordinate frames in various parts of the structure to visualize the rotation of the DH-coordinate frame or the simulated IMU sensors attached to the structure. An example of the visualization is presented in Figure 9.

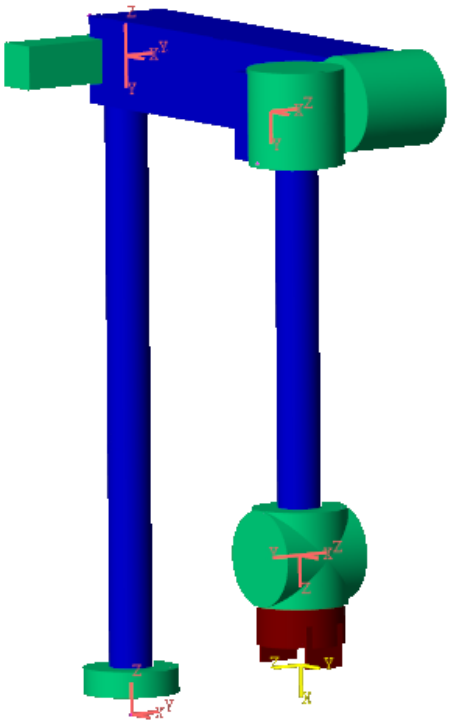

Figure 9. In Aaria's visualization, green cylinders and blocks present revolute and prismatic joints. Blue cylinders and blocks present links.

The possible manipulator configurations include structures similar to real physical robots. Most of the configurations do not closely resemble real robots and some would be useless in any real application. Despite that, these unconventional configurations can perform movements that are typical to robotic manipulators. This means that they can generate useful data just as well as any handcrafted configuration based on a real industrial manipulator. Additional examples of randomized structures are presented in Figure 10 and Figure 11.

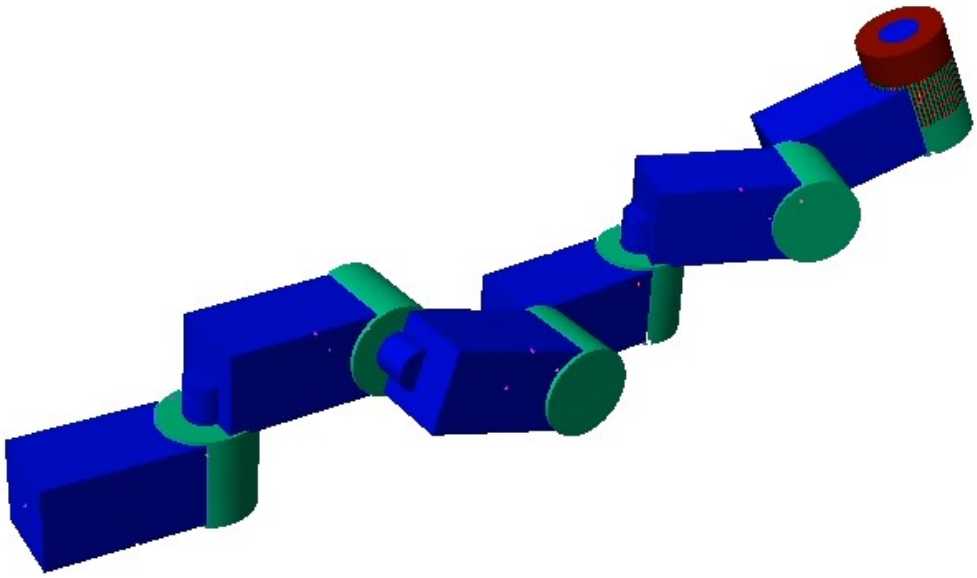

Figure 10. An example of a possible randomized configuration with 6 revolute joints.

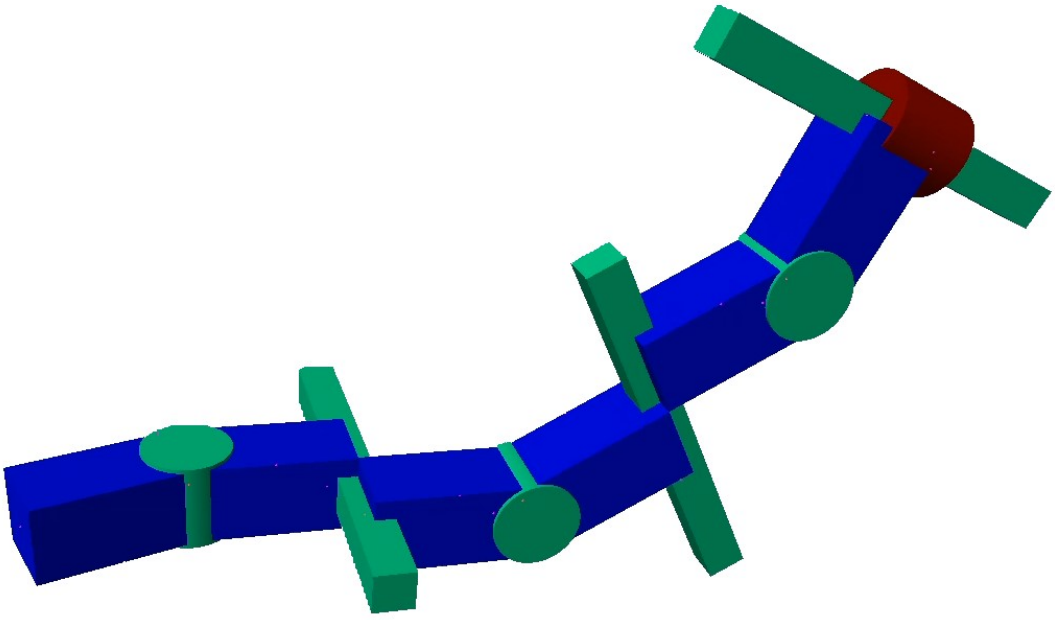

Figure 11. Another example of a randomized configuration with 3 revolute and prismatic joints.

There is an option to simulate a camera attached near the tool. The motivation for creating the camera functionality was to explore the possibilities of using Aaria to generate visual data. The camera model was attached next to the tool at an angle pointing towards the tooltip. The synthetic video data could be potentially used for learning item grasping but that was not a priority so the camera module was left out of the main model. The camera module was used in the joystick controlled model to help the user align the gripper with the target. The camera block and the video capture solution are part of Matlab and easy to add to a model.

3.8 Aaria's specific structure for the machine learning method

For the test project, we needed a very simple and specific structure. We used a single link manipulators made out of a revolute joint connected to a rigid base and a link with constant cross-section and variable length and density. The manipulators had to be controlled with different kinds of torque inputs to extract the required data from the simulation. We used three different torque profiles: rising ramp function, filtered step function and a sine wave. We wanted to collect data of joint position, velocity and acceleration, which was easy to do by utilizing the built-in features of the joint blocks. A more elegant and challenging approach would have been to use only noisy IMU data, but at this stage, joint movement outputs are used.

A simple way to implement a simulation where a structure is controlled by different torque profiles in a controlled and identical environment is to simulate them all at once. The manipulator was modelled as a set of three duplicated modules attached to the same base. After that, the torque input blocks were changed so that each copy of the manipulator had a different kind of torque input. All the manipulators are connected to stationary floating revolute joints that has $1\ \frac{Nm}{deg/s}$ internal damping. In the beginning of the simulation, all the manipulators are at the lowest position. The manipulators do not interact with each other in any way even though they can occupy the same space during the simulation. Running the simulations this way is beneficial because it removes the need of follow up simulations with same structure and different torque parameters. Another benefit is that the output of a simultaneous simulation is a single data and label table that can be easily converted into a dataset. A visualization of a simulation is presented in Figure 12.

To generate large amounts training data for the project, the simulation model was run in parallel by using parsim command of Matlab. This allows a separate simulation to be run on all the processor cores simultaneously, thus reducing the time to gather the required data. For some possibly hardware related reason the parallel simulations slow down over time and the simulations start initializing at the same time for all cores, which slows down the data gathering even further. To avoid these problems, we ran the simulations by using a two-loop structure, where the inner loop initializes and simulates 50 parallel simulations and the outer loop repeats this process an arbitrary amount of times and saves the outputs

after each 50 simulations. The last step in generating the final dataset is to combine all the simulation output files into a single data file and a label file.

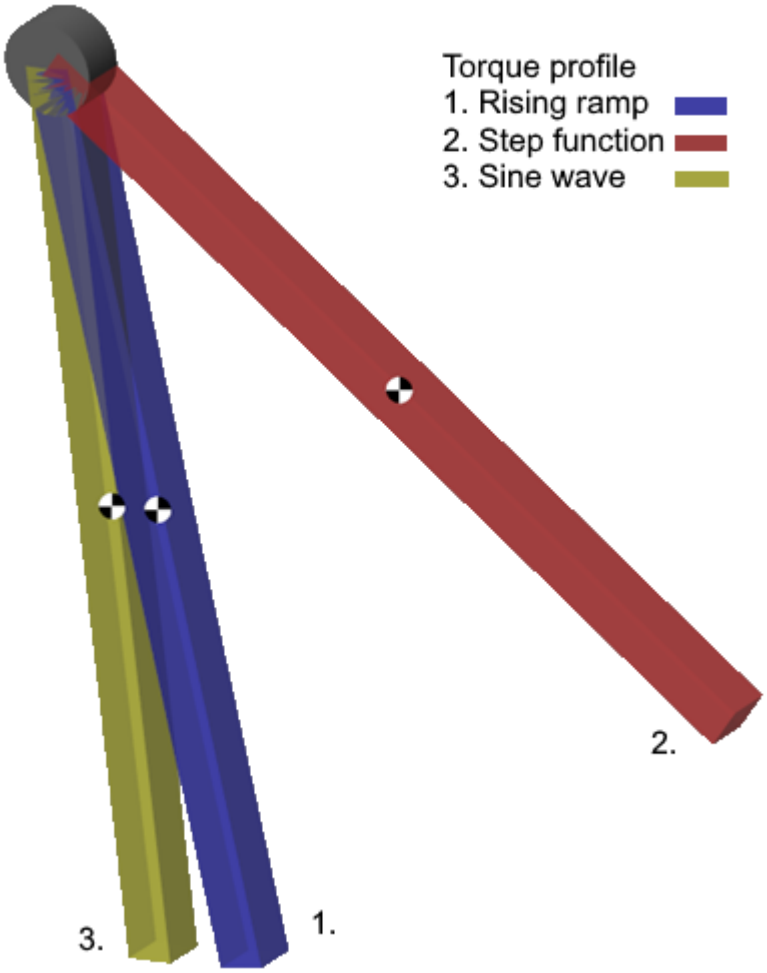

Figure 12. This model simulates three mechanically identical structures with different input torque profiles. Structures with different torque profiles are presented with color-coded and numbered beams.

At the beginning of each simulation, all the beams rest in a low position. Beam 1 is controlled by a rising ramp torque profile which causes the beam to slowly accelerate and rotate counter clockwise. Beam 2 is controlled by a filtered step function torque profile that steps up into full torque at 1-second mark and back down to zero torque at 6-second mark. Beam 3 is controlled by a sine wave torque profile. The sine wave outputs both positive and negative torque values and its frequency is 1 rad/s. The simulation scenario in Figure 12 presents a moment in a simulation around the 4-second mark. Beam 1 is affected by 40 % of the maximum torque and it is still near the bottom. Beam 2 has been controlled by 100 % torque for 3 seconds before this moment and it significantly ahead of beam 1. Beam 3 is controlled by negative 75 % torque and is currently rotating clockwise near the bottom.

Machine learning applications require large amounts of data. Data quantity is important, but so is data quality. ML models can only learn features that are present in the data so it is important that the training data include all the features we want the model to learn. We randomize the simulation parameters to get a reasonable variety of data. However, completely random values will not result in a useful dataset. The values need to be kept in a reasonable range considering the real world counterpart of the simulated structure. If the goal is to apply the results to manipulators that range from small robots to excavators, it is reasonable to limit the link length randomization to that scale. The dataset was created with link lengths that range from 20 cm to 2 meters. The mass of the links was manipulated by changing the density of the links. The average density was set to 1000 kg/m^3 and the upper and lower limits were set to half and double of the average so the possible link densities varied between 500 and 2000 kg/m^3. The last tunable parameter is the torque that controls the movement of the joint. The torque was randomized individually for each control trajectory. The randomization limits for torque are very important because too low torques fail to move the structure and the sensors cannot measure the movement. On the other hand, too high torque will spin the joints uncontrollably and the measurements may get distorted and the simulation might crash. Additionally, a real world robot would never be used thus way so simulating that is not reasonable. The sweet spot for torques was found experimentally to be between 50 and 500 Nm. Even the lowest torque can move the heaviest link and the highest torque will not spin the lightest link too fast. The torque values were used as the maximum and negative minimum value for the sine wave shaped control signal. The step function stepped from zero to the set torque value and then back to zero. The ramp signal was set to start from zero and increase 10 % of the torque value every second so it reaches the target value at the end of the simulation.

4. APPLICATION OF THE FRAME WORK IN MACHINE LEARNING

This chapter presents a ML application made by utilizing the synthetic data generated by the simulator from previous chapter. The best performance was achieved by using a Convolutional neural network (CNN) so this chapter starts with a brief description of them. After that, this chapter describes the entire process from data preprocessing to analyzing the outputs of a trained network.

4.1 Convolutional neural networks

CNNs are a popular and powerful tool for machine learning tasks such as image classification, voice recognition and natural language processing. LeCun et al. (1998) first introduced CNNs in their paper regarding document recognition. The roots of CNNs are in image processing but they can be used to process any kind of data as long as it can be presented in image-like format. The CNNs start the learning process by detecting low-level features. Deeper levels of the network combine those features and detect more complex features.

Convolutional neural networks consist of one or more convolutional layers. Neurons in convolutional layers are connected to a relatively small area of the input near their locations. Limiting connections from the input just to the nearby neurons significantly reduces the amount of needed parameters and memory. The CNN applies filters to the neurons which makes the features matching the filter's more visible. The same filters are applied to all the neurons in a layer so a feature that has been learned in one part of the input, can be detected in a different part of the input as well. Convolutional layers can be stacked on top of each other so that the neurons of the next layer are connected to the nearest outputs of the previous convolutional layer. The first layers learn and detect low level features and the later layers combine the previously detected features into more complex features.

Mathematically, the convolution operation is multiplication of matrixes (Goodfellow et al. 2016). Part of the input matrix is multiplied by the values in the kernel matrix. The kernel is a relatively small matrix compared to the input matrix. The kernel moves over the input and applies the convolution operation at regular intervals. This interaction is demonstrated in Figure 13, where 2x2 kernel applies convolution to the input matrix. In this example, the kernel moves one step at a time in both horizontal and vertical direction. The step size is called stride. When the stride is smaller than the kernel dimensions, parts of the input matrix are processed multiple times. In the example in Figure 13, input *f* is part of four convolution operations. The stride can also be set equal to the size of the kernel so each input will take part in the convolution exactly once. It is also possible to

set the stride higher than the kernel dimensions, which results in skipping of some inputs. Sometimes, usually near the edges of the input matrix, the kernel does not fit entirely within the input matrix. The convolution cannot be performed if some of the input values are missing. For example in Figure 13, moving the kernel down two steps puts the lower half of the kernel outside of the input matrix. This problem can be solved by simply not preforming these convolutions. This method is often referred as "valid" convolution. Another solution is to add zeroes around the input matrix and then perform "same" convolution using the added zeroes. The best method is usually somewhere between these two methods (Goodfellow et al. 2016).

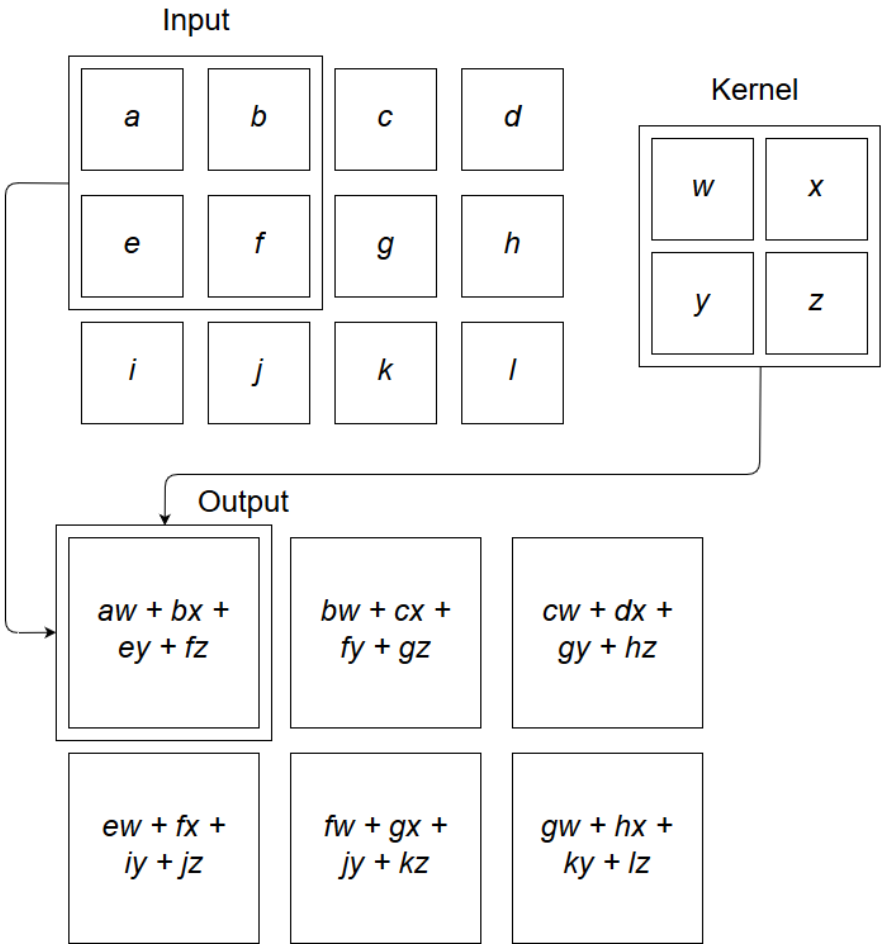

Figure 13. An example of 2D convolution. The figure is based on one found in (Goodfellow et al. 2016)

Another important part of CNNs is a pooling layer, which was also introduced by LeCun et al. in 1998. Pooling layers are connected to their inputs much like convolutional layers. A single neuron in a pooling layer is connected to the closest outputs of the previous layer. The neurons in pooling layers reduce their inputs to a single output by following a simple rule such as maximum or average of the input values. Pooling makes the network less sensitive to small translations in the input because most of the pool outputs will stay the same after a small translation (Goodfellow et al. 2016). The amount of pooling units can be set equal to the inputs to maintain the matrix dimensions. It is also possible to have

less pooling units so that the pooling layers perform downsampling. Downsampling the layer outputs causes some information loss but it also benefits the network by reducing the computational and memory requirements by shrinking the size of the output. Examples of max pooling with and without downsampling are presented in Figure 14.

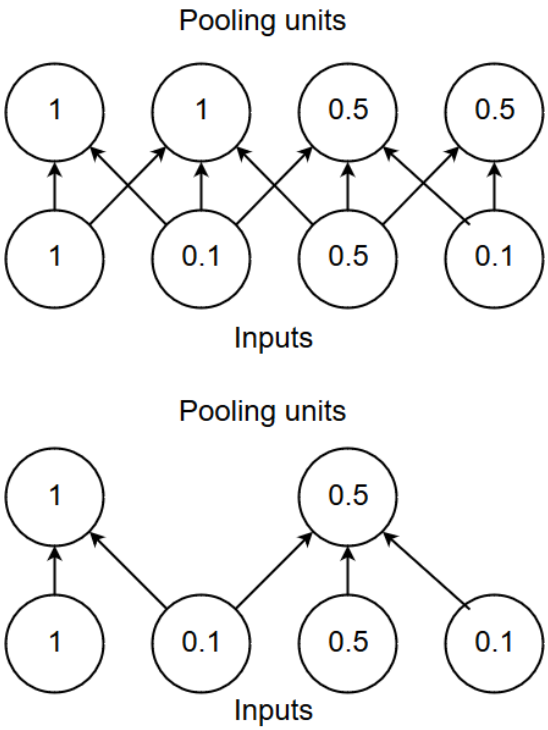

Figure 14. Max pooling preserves the input dimensions in the top example. The bottom example demonstrates max pooling with downsampling. Figure adapted from (Goodfellow et al. 2016).

During the training of a network, the process where inputs travel forward through the network and eventually produce an output is called forward propagation (Goodfellow et al. 2016). The reverse process of this is called back-propagation, where information travels to the opposite direction. Back-propagation computes a gradient that is used by optimization algorithms to modify the weights of the layers and make the network learn. Stochastic gradient descent is a popular optimization algorithm. It follows the descending gradient one small step at a time until the gradient reaches zero and the cost function is minimized (Géron 2017).

4.2 Preprocessing

The very first data preprocessing steps in this ML application were done in MATLAB. The simulation data was combined into a single 3D data array and a matching 2D label array. Each training instance contains a table with 201 rows and 12 columns. The first four columns have values for torque, rotation, angular velocity and angular acceleration when using the ramp shaped torque. The next four columns have the same measurements for the structure controlled with step function, and the last four columns are for the sine

wave controlled structure. Each row has a measurement taken every 0.05 seconds over a 10-second simulation. The arrays are combined into a big list of arrays. Each simulation has three labels. Link length, link mass and their multiplication. The labels are combined into a big list just like the data arrays so that they have matching indexes. Both lists are then saved as an h5 array.

The following preprocessing steps were done in Python by using Scikit-Learn (Pedregosa et al. 2011). The data was first loaded in and saved as data and labels by using h5py package. Next, the data and labels were transposed with the help of numpy so that array dimensions were sorted in descending order. Next, the dataset was split into training and test sets. The labels of the datasets were not processed in any way and the training and test set data were processed separately but in the same way. Next, the data was standardized and scaled. Unfortunately, the scalers do not support 3D arrays, so first we had to reshape them into a long 2D array where each feature was in their own column and all the instances were listed back to back. We tried using Scikit-Learn's Standard Scaler to subtract the mean and to scale the values to unit variance but this did not improve the performance of the network. Next, we used Min Max scaler to scale the values to range from -1 to 1. After the scaling, the data was reshaped back into the original 3D format and a fourth dimension was added because the data will be fed into a convolutional neural network which expects the last dimension to be channels. In this case, there is only one channel but it is possible reshape the data so that the data from each separate control method is in their own channel. It is an interesting idea but it did not perform well in practice. After these steps, the data is ready to be fed into the ML model.

4.3 Machine learning model

The ML portion of this project was done with Keras application programming interface (API) (Chollet et al. 2015) and by using TensorFlow (Abadi et al. 2015) backend. TensorFlow is an open source machine-learning framework by Google. Keras API simplifies the process of building neural networks by offering a more user-friendly way to write code. The combination of Keras' high popularity and plain TensorFlow code being rather clunky has resulted in TensorFlow team's decision to integrate Keras in the upcoming TensorFlow 2.0.

4.3.1 Network architecture

The CNN architecture for this task went through many iterations while looking for the optimal structure to maximize the efficiency and accuracy. Figure 15 presents a sample structure of a relatively simple CNN that still performs quite well.

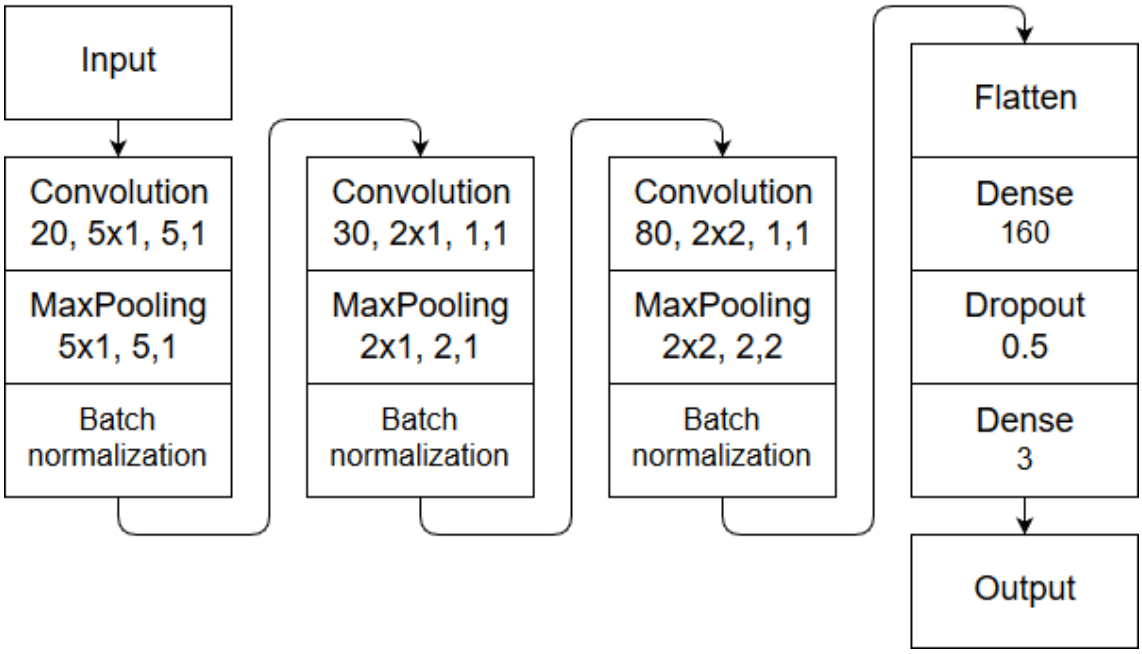

Figure 15. A simple diagram of the CNN architecture used in the task

The first layer of the CNN is a 1D convolutional layer where each neuron is connected to a vertical line of five inputs in a single column. The strides are set to same value as the kernel dimensions so every input is processed exactly once. All the convolutional layers use valid padding which means that the parts of the input that do not fit inside the convolutional kernels will be ignored. Another option would be to use zero padding where zeroes would be added around the input so that all of the input data fits inside the kernels. The valid padding ignores the last value of the 201 input values in each column. This layer uses a rectified linear unit (ReLU) activation function, which is commonly used in CNNs. The next layer is a max pooling layer with the same pool size as the kernel in the previous convolutional layer. Next, a batch normalization layer is placed to improve the performance of the network. Batch normalization layer subtracts the mean and divides by standard deviation and this way normalizes the output of the previous layer. The reason why batch normalization improves the performance of the networks was believed to be the reduction of the internal covariate shift. However, recent paper by (Santurkar et al. 2018) disagrees with that belief and suggests that batch normalization improves the stability of the underlying optimization problem and makes the gradients behave better.

The next layer is another convolutional layer. This time, the amount of filters is increased from 20 to 30 and the kernel size is reduced from 5 to 2. The stride of the kernel is reduced to one, which means that each input is processed twice as the 2 units long kernel passes over them. The width of the kernel remains 1 to keep the operation contained in a single column. CNNs work best on data where the neighboring data points tend to correlate. Consecutive measurements in a single column tend to be close to each other but this relationship cannot be expected from different columns with different measurements. The

convolutional layer is once again followed by pooling layer with matching sized pool and batch normalization layer.

The last convolutional layer uses a 2x2 kernel with 80 filters, which mixes features of two neighboring columns. Doing this is a necessary action to extract the information hidden in the relationship of the different measurements. However, the exact reason why and how this works is not clear. CNNs work well for images because each pixel has a meaningful relationship to the neighboring pixels. In time series, neighboring values in the same column have similar relationship as in pictures. However, values in different columns do not have such a relationship. This means that the order of columns can be shuffled and the data is still in readable format unlike pictures. This also means that the seemingly meaningless order of the columns does affect the outcome of the convolution operation when the kernel is wider than the column.

The last block of the network contains a flatten layer which flattens the input and prepares it for the dense layer. The dense layer is a simple fully connected neural network layer that routes the signals from the CNN to the output. A dropout layer randomly removes half of the connections going through it This regularizes the network and reduces overfitting. The output layer in this network is dense layer with three neurons because this network performs three regression tasks simultaneously.

4.3.2 Training

During the training, the loss function compares the output of the network to the labels. In this regression task, we use mean absolute error (MAE) loss function. The network makes changes to itself trying to reduce the value of the loss function. According to Géron (2017) the mean absolute error can be expressed as:

$$MAE(X, h) = \frac{1}{m} \sum_{i=1}^{m} \left| h\left(x^{(i)}\right) - y^{(i)} \right| \quad (1)$$

Where X is the matrix of all feature values, h is the prediction function of the system, i is the index of an instance in the dataset, m is the number of instances in the dataset, x is a single instance in the dataset specified by index i and y is the corresponding label with a matching index.

The optimizer controls the way the network changes its gradients. This network uses Adadelta optimizer, which adapts learning rates based on the most recent gradient updates. Learning rate is one of the most important parameters during training. Too low learning rate slows down the convergence of the algorithm and too high learning rate can cause divergence of the algorithm.

The duration of the training process is determined by the network size, amount of training data and the number of epochs. An epoch describes a phase during the training where all

of the training data is processed once. The weights of the network continue to update during the epochs until the training finishes the last planned epoch. The network may converge before the last epoch and any training done after that does not produce any additional benefits. Early stopping is a method where the network performance is monitored during the training and the training is automatically stopped when the performance of the network no longer improves. Early stopping speeds up the training time by removing the redundant computation and it also prevents possible overfitting.

Training this network with 360000 training instances takes around 25 minutes with NVIDIA Quadro P600 GPU and Intel Core i7-7700 CPU. During and after the training, the network's performance is monitored by predicting the values of validation dataset. The validation dataset is a part of the initial dataset that was not used in training. The network can be expected to work about as well on new data as it performs on the validation dataset.

The progression of the training can be monitored by looking at the output of the python code. The output can be customized to show useful metrics like loss and accuracy for both the training set and the validation set. Another way to monitor the training is by using TensorBoard. TensorBoard is a visualization tool that can plot graphs of the important metrics and visualize the network structure among other things. An example of a TensorBoard visualization can be seen in Figure 16. The red line presents the previously described network.

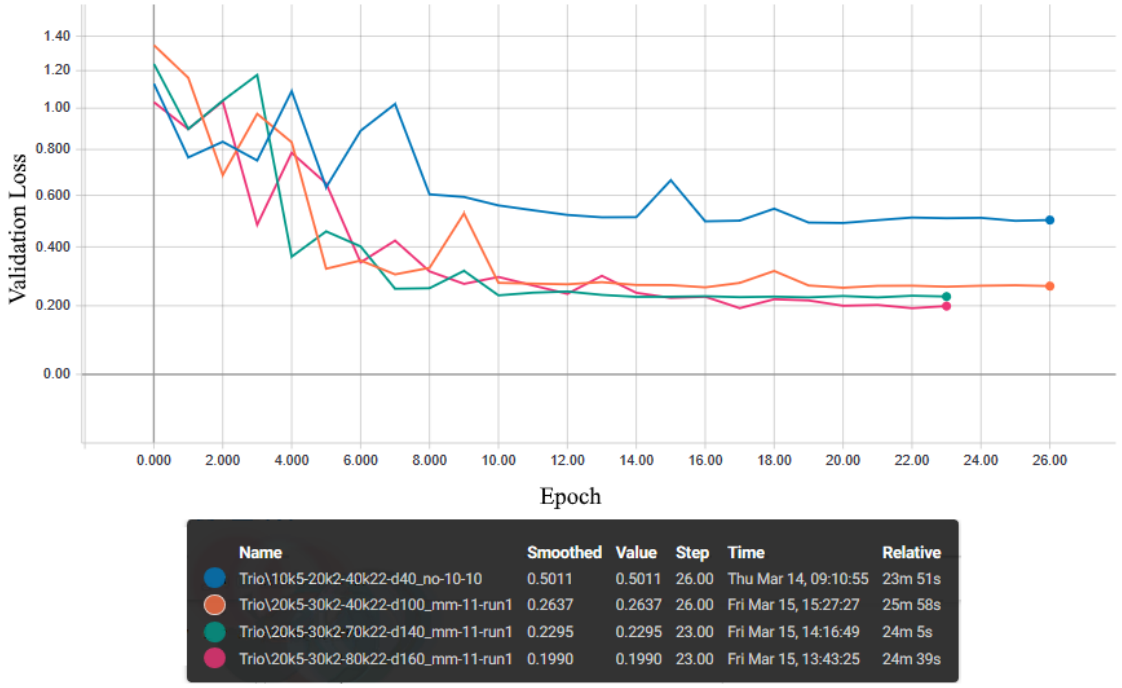

Figure 16. An example of TensorBoard displaying the validation loss of four different models during the training.

4.3.3 Prediction and performance analysis

There are many ways of analyzing model performance. The type of the task determines what kind of performance measures are reasonable to use. Classification performance is often measured with metrics such as accuracy, precision and recall. Accuracy simply describes how large fraction of the predictions was correct. Precision measures the fraction of true positive classifications out of all positive classifications. Whereas recall measures the ratio of true positive classifications and all positive instances (Géron 2017). Accuracy is a simple metric but it may not tell the whole truth about how well the system performs. For example, a system that detects all cases of a rare cancer but also gives a lot false positive detections is considered better than a similar system that only detects some of the cases but gives no false positives. In this case, a system with low accuracy and high recall is better than a system with high accuracy and high precision. These are just a few examples of the various performance measures used in classification tasks to determine the performance in the most meaningful way.

Regression metrics have less variety than classification metrics. The most important component of these metrics is the numerical difference between the true value and the prediction called error. The error can be presented in many forms such as mean absolute error, mean squared error and median absolute error. These metrics primarily describe how close the predictions are the true value. There are other metrics such as explained variance score and R^2 score (Pedregosa et al. 2011). These metrics describe the overall goodness of the model with a score from negative values up to one. These are quite common metrics and additional metrics can be handcrafted to gain additional insight into the model's performance.

The parameters for the task were chosen so that the length of the structures ranged from 0.2 m to 2 m. The randomized structure density set the masses of the structures to a range from 1 kg to 40 kg. The average and maximum absolute prediction errors on the validation set for each property are presented in Table 1. Mean relative errors are calculated by dividing each absolute error value by the ground truth and calculating mean of those values. Mean relative errors are also presented in Table 1 along with explained variance scores calculated by a function from Scikit-Learn (Pedregosa et al. 2011).

Since the dynamic effects of changes in mass and its distance from the axis of the joint can have interference, their multiplication could have better error magnitudes. Note that, individual mass of a structure, independent of the length, is considered unobservable in classic observation methods.

Table 1. Prediction errors on validation set.

Link feature	Mean absolute error	Mean relative error	Maximum absolute error	Explained variance score
Length	0.02 m	2.5 %	0.3 m	0.9961
Mass	0.19 kg	2.5 %	1.0 kg	0.9994
Length*Mass	0.08 kg*m	4.1 %	2.8 kg*m	0.9999

Each prediction task is also presented in a graph that shows true data point in blue and predicted values in red. The predicted value is presented on the vertical axis and the horizontal axis simply shows the amount of data points. The data points are sorted by the value of validation data to make the graphs easier to read. This also reveals some interesting characteristics the predictions and the validation data. Link length prediction is the least accurate of the three features based on explained variance score. The true length values form a smooth ramp unlike the prediction that switches from side to side giving high predictions on the lower end of the scale and low predictions on most of the higher half of the data. In addition, the predictions tend to exaggerate the values in both ends of the scale. The mean relative prediction error was 2.5 %. The link length data and predictions are shown in Figure 17.

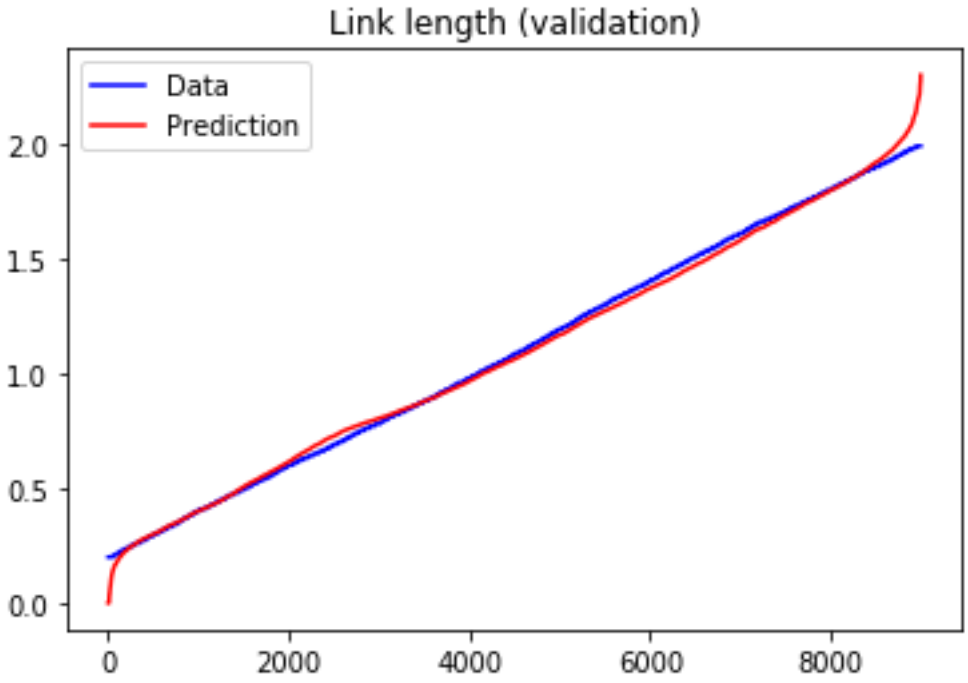

Figure 17. Link length values in validation set and their corresponding predictions.

The link mass predictions are quite accurate. The mean absolute error was only 0.19 kg and even the maximum error was 1.0 kg. The curves on the graph can be described as rising slopes that curve upwards towards the end. The shape is explained by the fact that link mass is determined by both the length and the density of the link. The high end curves upwards rapidly because the mass is being increased by high values of both, length and density. A combination of a high and a low value or two average values lands somewhere on the linear part of the graph. The combination of two high values will result in a mass that is on the curved part of the graph. The prediction follows the shape of the data curve reasonably well but it does show similar side-to-side switching as the length prediction graph. The mean relative prediction error was 2.5 %. The data and predictions of link mass are presented in Figure 18.

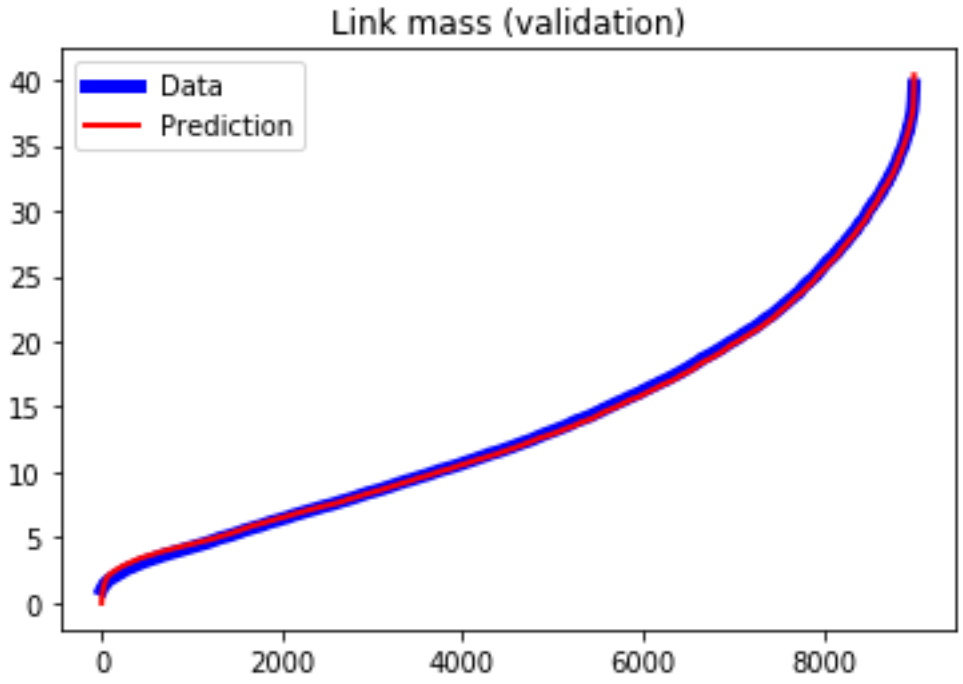

Figure 18. Link mass values in validation set and their corresponding predictions.

The third prediction was done on the multiplication of link length and mass. The goal was to make an easier feature to predict by eliminating the possible confusion between short and heavy links and long and light links that require similar torques to move. The shape of the graph looks similar to the mass prediction graph. The prediction and data curves are overlapping more evenly than in the previous graphs. However, the mean relative error is 4.1 %, which suggests that the prediction as a whole is less accurate than the other predictions. On the other hand, the explained variance score of this prediction is the highest so declaring the overall most predictable feature is a matter of aspect. The data and predictions of the multiplication of link mass and length are presented in Figure 19.

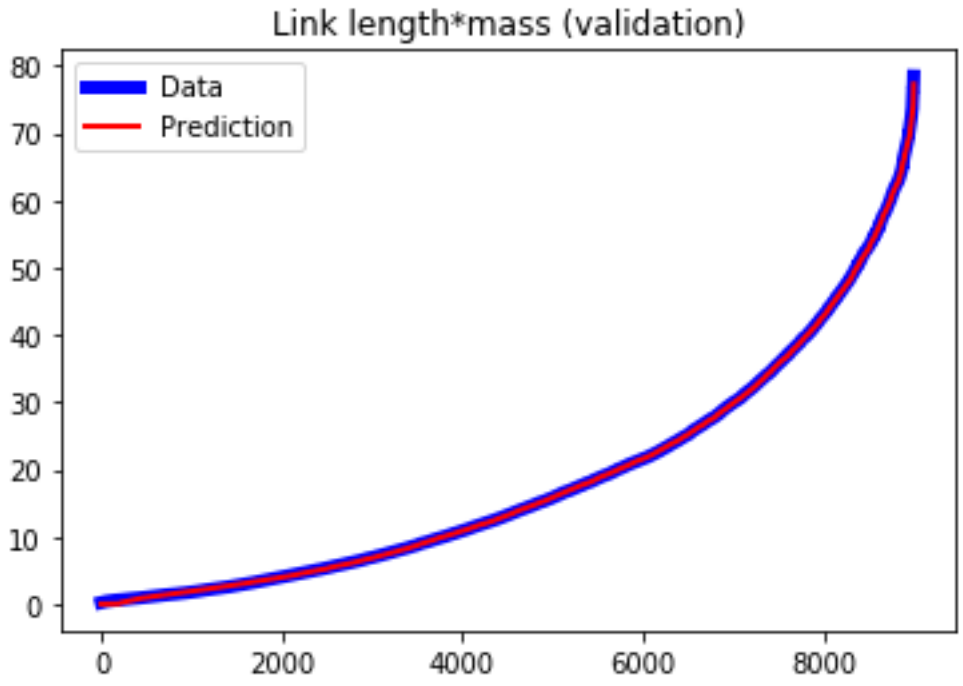

Figure 19. Data and predictions of multiplication of link mass and length.

The task was quite simple from a computational perspective. Most of the models ran for about 25 minutes until early stopping callback function stopped them. Some slightly larger models ran over an hour but the additional complexity and processing time did not result in significant performance gains. The model could benefit from additional training data but increasing the amount of training data also increases the memory requirement. Upgrading the memory and training with a bigger dataset will increase the training time unless the GPU is upgraded as well. The performance of the network could be increased with more training data and with a better computer but at this stage the goal is more of a proof of concept than a final product so additional performance is not required. A good alternative for a computer upgrade is the use of cloud-computing services.

A trained model will most likely perform worse in real applications than it performs in the testing stage (Géron 2017). In most cases, even a large dataset cannot include every possible feature the model will encounter. In addition, training with synthetic data further reduces the performance due to the difference between real and synthetic data, which is often referred as the domain gap (Hinterstoisser 2017). The model would most likely experience decreased performance in a physical real world test setup. There are domain adaptation methods designed to mitigate the effects of domain gap. Usually these methods require some real measured data like in (Rad 2018) and (Tercan 2018). We do not have a physical test setup for gathering real data for this task so we are unable to evaluate the model's performance on real data.

5. CONCLUSION

Aaria is a simulation model for generating synthetic IMU data for robotic ML applications. The modular structure and reconfigurability enables Aaria to generate randomized structures and trajectories for manipulators. This is important for generating data with sufficient variety for ML applications. The manipulator configurations are created based on modified DH-parameters, which makes it compatible with numerous manipulator designs. Aaria can also be manually configured to a specific structures and trajectories if the use case so requires. The simulator was built in Matlab Simulink environment with blocks from Simscape Multibody library.

One of Aaria's biggest assets is the flexibility of the configurability, but it also causes one of its flaws. The configuration requires extensive parameter tuning. The first part of the parameter tuning is to find the right parameters to tune to get the wanted effect and the second part is to find the suitable values for the parameters. While neither of these tasks is overwhelmingly difficult, there is room for user error. Matlab Simulink environment offers functional tools for creating and running simulations but on the other hand, that also limits the accessibility of Aaria by requiring users to have an active Matlab license.

Aaria can be used to generate data for many kinds or robotic ML applications. The goals of the task determine the parameters and possibly needed modifications for Aaria to generate the required data. Based on the requirements, Aaria can be parametrized to generate suitable data. Once the data is gathered, it can be used to train several different models to find the best model. Some preprocessing is usually beneficial for the performance of the models. Feature engineering may be used in addition to the usual normalizing and scaling. CNNs will find features by themselves but converting the data to a more meaningful format may improve the performance of the models. The best models can be then further optimized by hyperparameter tuning. A few good models can be combined into an ensemble that will probably perform slightly better than any single model.

The ML application presented in this work utilizes data from a modified Aaria model to predict lengths and masses of rotating structures based on input torques and movement measurements. The solution uses a CNN to extract the features from the data and to solve the regression task. The results of the regression task show that the CNN is able to predict the values for lengths and masses separately with high accuracy despite the fact that these features have a similar effect on the torque-controlled movement. This ML application could be developed further by creating an ensemble model to possibly increase the regression performance. Another path of further development is applying the ML application to a physical test setup to bridge the gap between synthetic and measured data.

REFERENCES

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man´e, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Van-houcke, V. Vasudevan, F. Vi´egas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, (2015) Tensor-Flow: Large-scale machine learning on heterogeneous systems, Software available from tensorflow.org

M. A. Ahandani, A. R. Ghiasi, H. Kharrati, (2018) Parameter identification of chaotic systems using a shuffled backtracking search optimization algorithm, Soft Computing, December 2018, Volume 22, Issue 24, pp 8317–8339

Analog Devices. Available: https://www.analog.com/en/products/adis16485.html

J. Bjurgert, P. E. Valenzuela and C. R. Rojas, (2018) On Adaptive Boosting for System Identification, IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 9, pp. 4510-4514

C. Caramia, D. Torricelli, M. Schmid, A. Muñoz-Gonzalez, J. Gonzalez-Vargas, F. Grandas, J. L. Pons, (2018) IMU-Based Classification of Parkinson's Disease From Gait: A Sensitivity Analysis on Sensor Location and Feature Selection, IEEE Journal of Biomedical and Health Informatics, volume 22, pages1765-1774

G. Čepon, J. Rogelj, L. Knez, M. Boltežar, (2018) On multibody-system equilibrium-point selection during joint-parameter identification: A numerical and experimental analysis, Mechanism and Machine Theory, Volume 128, October 2018, pp. 287-297

F. Chollet et al. (2015) Keras, Available: https://keras.io

J. J. Craig, (1986) Introduction to Robotics: Mechanics & Control, Addison-Wesley Publ.

J. Davey, N. Kwok, M. Yim, (2012) Emulating self-reconfigurable robots – design of the smores system, Intelligent robots and systems (iros), 2012 ieee/rsj international conference on, Vilamoura, Algarve, Portugal, pp. 4464-4469.

H. M. Do, G. H. Kim, T. Choi, D. H. Kim, Y. Son, (2016) Development of simulation model for modular and reconfigurable robots, 2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence *(URAI)*, Xi'an, pp. 138-139

G. Gao, G. Sun, J. Na, Y. Guo, X. Wu, (2018) Structural parameter identification for 6 DOF industrial robots, Mechanical Systems and Signal Processing, Volume 113, pp. 145-155

Gartner, (2018) Available: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/, accessed 15.01.2019

Gartner, (2019) Available: https://www.gartner.com/it-glossary/big-data/, accessed 11.03.2019

A. Géron, (2017) Hands-On Machine Learning with Scikit-Learn & TensorFlow, O'Reilly Media

I. Goodfellow, Y. Bengio, A. Courville, (2016) Deep Learning, MIT Press

A. Hanel, D. Kreuzpaintner, and U. Stilla, (2018) Evaluation of a Traffic Sign Detector by Synthetic Image Data for Advanced Driver Assistance Systems, Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLII-2, 425-432

A. Hautakoski, M. M. Aref, J. Mattila, (2018) Reconfigurable Manipulator Simulation for Robotics and Multimodal Machine Learning Application: Aaria

S. Hinterstoisser, V. Lepetit, P. Wohlhart, K. Konolige, (2017) On Pre-Trained Image Features and Synthetic Images for Deep Learning, arXiv Preprint

J. Hoffmann, Y. Bar-Sinai, L. Lee, J. Andrejevic, S. Mishra, S. M. Rubinstein, C. H. Rycroft, (2019) Machine Learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets, eprint arXiv:1807.01437

S. Hong, D. Choi, S. Kang, H. Lee and W. Lee, (2016) Design of manually reconfigurable modular manipulator with three revolute joints and links, 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, pp. 5210-5215

D. Horn, S. Houben, (2018) Evaluation of Synthetic Video Data in Machine Learning Approaches for Parking Space Classification, 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, pp. 2157-2162.

Y. Hsu, S. Yang, H. Chang, H. Lai, (2018) Human Daily and Sport Activity Recognition Using a Wearable Inertial Sensor Network, IEEE Access, vol. 6, pp. 31715-31728

G, Jing, T, Tosun, M, Yim, H. Kress-Gazit, (2018) Accomplishing high-level tasks with modular robots, Autonomous Robots 2018, Volume 42, Issue 7, pp 1337–1354

D. Jung, J. Cheong, D. I. Park, C. Park, (2018) Backward sequential approach for dynamic parameter identification of robot manipulators, 2018 International Journal of Advanced Robotic Systems

P. Kasnesis, C. Z. Patrikakis, I. S. Venieris, (2018) PerceptionNet: A Deep Convolutional Neural Network for Late Sensor Fusion, Intelligent Systems Conference 2018

J. A. Kereluk, M. R. Emami, (2015) A New Modular, Autonomously Reconfigurable Manipulator Platform, International Journal of Advanced Robotic Systems

Y. LeCun, Y. Bengio, G. Hinton, (2015) Deep Learning, Nature 521, pp. 436-444

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, (1998) Gradient-based learning applied to document recognition, Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324

C. Liu, W. Hsaio, Y. Tu, (2018) Time Series Classification with Multivariate Convolutional Neural Network, IEEE Transactions on Industrial Electronics, Volume: 66, Issue: 6, June 2019, pp. 4788 - 4797

D. Malmgren-Hansen, A. Kusk, J. Dall, A. A. Nielsen, R. Engholm, H. Skriver, (2017) Improving SAR Automatic Target Recognition Models With Transfer Learning From Simulated Data, in IEEE Geoscience and Remote Sensing Letters, vol. 14, no. 9, pp. 1484-1488

A. Marcu, D. Costea, V. Licăreţ, M. Pîrvu, E. Sluşanschi, M. Leordeanu, (2019) SafeUAV: Learning to Estimate Depth and Safe Landing Areas for UAVs from Synthetic Data, In: Leal-Taixé L., Roth S. (eds) Computer Vision – ECCV 2018 Workshops. ECCV 2018. Lecture Notes in Computer Science, vol 11130, pp. 43-58, Springer, Cham

J. Margarito, R. Helaoui, A. M. Bianchi, F. Sartor, A. G. Bonomi, (2016) User-Independent Recognition of Sports Activities From a Single Wrist-Worn Accelerometer: A Template-Matching-Based Approach, in IEEE Transactions on Biomedical Engineering, vol. 63, no. 4, pp. 788-796

J. McCarthy, (2007) What is Artificial Intelligence, Available: http://www-formal.stanford.edu/jmc/whatisai/node1.html

Mathworks, (2018) Simscape Multibody User's Guide

T. Mitchell, (2006) The Discipline of Machine Learning

F. J. Ordóñez, D. Roggen, (2016) Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition, MDPI, Sensors 2016

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, É. Duchesnay, (2011) Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, volume 12

N. Puri, (2017) Inverse Kinematics on Kuka Arm using ROS and Python, Available: https://nitishpuri.github.io/posts/robotics/inverse-kinematics-on-kuka-arm-using-ros-and-python/

M. Rad, M. Oberweger, V. Lepetit, (2018) Feature Mapping for Learning Fast and Accurate 3D Pose Inference from Synthetic Images, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4663-4672

D. Roggen, A. Calatroni, M. Rossi, T. Holleczek, K. Förster, G. Tröster, P. Lukowicz, D. Bannach, G. Pirkl, A. Ferscha, et al., (2010) Collecting complex activity data sets in highly rich networked sensor environments. In Proceedings of the 7th IEEE International Conference on Networked Sensing Systems (INSS), Kassel, Germany, 15–18 June 2010; pp. 233–240.

C. A. Ronao, S. Cho, (2016) Human activity recognition with smartphone sensors using deep learning neural networks, Expert Systems with Applications Volume 59, pp. 235-244

C. Sakaridis, D. Dai, S. Hecker, L. Van Gool, (2018) Model Adaptation with Synthetic and Real Data for Semantic Dense Foggy Scene Understanding, The European Conference on Computer Vision (ECCV), pp. 687-704

S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, (2018) How Does Batch Normalization Help Optimization?, Advances in Neural Information Processing Systems 31 (NIPS 2018) pre-proceedings

C, Sobie, C. Freitas, M. Nicolai, (2018) Simulation-driven machine learning: Bearing fault classification, Mechanical Systems and Signal Processing, Volume 99, pp. 403-419

H. Tercan, A. Guajardo, J. Heinisch, T. Thiele, C. Hopmann, T. Meisen, (2018) Transfer-Learning: Bridging the Gap between Real and Simulation Data for Machine Learning in Injection Molding, 51st CIRP Conference on Manufacturing Systems, Volume 72, pp. 185-190

C. Viegas, M. Tavakoli, A. T. de Almeida (2017) A novel grid-based reconfigurable spatial parallel mechanism with large workspace, Mechanism and Machine Theory, Volume 115, pp. 149-167

X. Wang, F. Bi, H. Du, (2018) Reduction of low frequency vibration of truck driver and seating system through system parameter identification, sensitivity analysis and active control, Mechanical Systems and Signal Processing, Volume 105, pp. 16-35

Y. Yue, Y. Li, K. Yi, Z. Wu, (2018) Synthetic Data Approach for Classification and Regression, 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP), Milan, pp. 1-8

L. Zhang, A. Gonzalez-Garcia, J. van de Weijer, M. Danelljan, F. S. Khan, (2019) Synthetic Data Generation for End-to-End Thermal Infrared Tracking, IEEE Transactions on Image Processing, vol. 28, no. 4, pp. 1837-1850

T. Zhang, W. Zhang, M. M. Gupta, (2018) An underactuated self-reconfigurable robot and the reconfiguration evolution, Mechanism and Machine Theory, Volume 124 pp. 248-258

Y. Zhang, G. Song, S. Liu, G. Qiao, J. Zhang, H. Sun, (2016) A Modular Self-Reconfigurable Robot with Enhanced Locomotion Performances: Design, Modeling, Simulations, and Experiments, Journal of Intelligent & Robotic Systems, Volume 81, Issue 3–4, pp. 377–393

T. Zimmermann, B. Taetz, G. Bleser, (2018) IMU-to-Segment Assignment and Orientation Alignment for the Lower Body Using Deep Learning, Sensors 2018, 18, 302

APPENDIX A: AARIA MODEL OVERVIEW